Хмельницький національний університет
Факультет програмування та комп'ютерних і телекомунікаційних систем
Кафедра комп'ютерних наук та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему *Автоматизована система допомоги в діагностуванні хвороб терапевтом*

Галузь знань *12 – Інформаційні технології*
Шифр і назва галузі знань

Спеціальність *122 – Комп'ютерні науки*
Шифр і назва спеціальності

Освітня програма *Комп'ютерні науки*
Назва освітньої програми

Виконав: *студент 4 курсу, група КН-17-2* _____ *І.О. Вишинський*
Курс, група виконавця | Підпис | Ініціали, прізвище

Керівник: *к.т.н., доцент кафедри КНІТ* _____ *О.В. Мазурець*
Науковий ступінь, посада | Підпис | Ініціали, прізвище

Нормоконтроль: *к.т.н., доцент кафедри КНІТ* _____ *Р.О. Багрій*
Науковий ступінь, посада | Підпис | Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КНІТ, д.т.н., професор _____ *О.В. Бармак*
Підпис | Ініціали, прізвище

08 червня 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет *програмування та комп'ютерних і телекомунікаційних систем*
Кафедра *комп'ютерних наук та інформаційних технологій*
Освітній ступінь *бакалавр*
Галузь знань *12 – Інформаційні технології*
Спеціальність *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ
Завідувач кафедри *комп'ютерних наук та інформаційних технологій*

(підпис)
д.т.н., професор О.В. Бармак
«*08*» *травня* *2021* року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

1. Тема кваліфікаційної роботи бакалавра: *«Автоматизована система допомоги в діагностуванні хвороб терапевтом»*

2. Завдання видано студенту *Вишинському Іллі Олександровичу*
(прізвище, ім'я, по батькові)

3. Керівник роботи *доцент кафедри КНІТ Мазурець Олександр Вікторович*
(посада, прізвище, ім'я, по батькові)

4. Затверджено наказом університету від «*05*» *02* *2021* р. № *11*

5. Зміст пояснювальної записки (перелік задач) та вихідні дані:

Мета роботи – розробка автоматизованої системи допомоги в діагностуванні хвороб терапевтом. При апріорному формуванні діагнозу слід враховувати відповідності симптомів певним хворобам. Слід забезпечити виконання функцій обліку даних про пацієнтів, клініки, лікарів, записування візитів; отримання статистики та отримання приблизного діагнозу на базі заданих симптомів, й забезпечити одержання результату роботи системи у вигляді автоматизації дій терапевта поліклініки, зокрема запису візитів пацієнта, обліку даних пацієнтів, призначення діагнозів та лікувань тощо.

Виконавець: *студент 4 курсу, група КН-17-2* (Курс, група виконавця) _____ (Підпис) *І.О. Вишинський* (Ініціали, прізвище)

Керівник: *к.т.н., доцент кафедри КНІТ* (Науковий ступінь, посада) _____ (Підпис) *О.В. Мазурець* (Ініціали, прізвище)

Анотація

Тема кваліфікаційної роботи бакалавра: *«Автоматизована система допомоги в діагностуванні хвороб терапевтом»*

Виконавець кваліфікаційної роботи бакалавра: *студент групи КН-17-2 Вишинський Ілля Олександрович*

Керівник кваліфікаційної роботи бакалавра: *к.т.н., доцент кафедри КНІТ Мазурець Олександр Вікторович*

Кваліфікаційна робота бакалавра містить:

Пояснювальна записка				Кількість додатків
Сторінок	Рисунків	Таблиць	Джерел інформації	
57	44	21	48	5

Метою кваліфікаційної роботи бакалавра є створення автоматизованої системи допомоги в діагностуванні хвороби терапевтом. Для розробки інформаційної системи було використано мову програмування JavaScript, фреймворки Express, React, Redux та систему керування базами даних MySQL.

Розроблена система призначена для терапевтів в поліклініках. Реалізована автоматизація обліку даних про пацієнтів, клініки, лікарів. Автоматизований процес реєстрації візитів, отримання статистики та отримання приблизного діагнозу на базі заданих симптомів.

Напрямками практичного використання розробленої інформаційної системи визначено автоматизацію дій терапевта поліклініки, таких як: запис візиту пацієнта, облік даних пацієнта, призначення діагнозу та лікування.

Ключові слова: терапевт, поліклініка, інформаційна система, діагностика, візит, хвороба, діагноз.

Виконавець: *студент 4 курсу, група КН-17-2* (Курс, група виконавця) _____ (Підпис) *І.О. Вишинський* (Ініціали, прізвище)

Зміст

Перелік скорочень

Скорочення, термін, позначення	Пояснення
PHP	Hypertext Preprocessor
БД	База даних
API	Application Programming Interface
ІТ	Інформаційні технології
КН	Комп’ютерні науки
КНІТ	Комп’ютерні науки і інформаційні технології
ПЗ	Пояснювальна записка
СКБД	Система керування базами даних
ІС	Інформаційна система
КРБ	Кваліфікаційна робота бакалавра
CLR	Common Language Runtime
ОС	Операційна Система
HTML	HyperText Markup Language – «мова гіпертекстової розмітки»
CSS	Cascading Style Sheets
JS	JavaScript
MS	Microsoft
SQL	Structured Query Language
ЦБД	Центральна база даних
REST	Representational State Transfer
JSON	JavaScript Object Notation
PHP	Hypertext Preprocessor
БД	База даних
API	Application Programming Interface

Вступ

Сучасна медицина в Україні має багато проблем. Однією з основних є велика завантаженість лікарів. У одного лікаря так багато пацієнтів, що одному з них він може приділити в середньому 15 хвилин за прийом. В багатьох випадках цього буває мало. Крім великої кількості пацієнтів, лікар також має багато паперової роботи, пов’язаної з заповненням карток хворих, виписування довідок тощо.

Також іноді виникають проблеми з діагностуванням хвороби лікарем. Лікар, як і будь-яка інша людина, може помилятись. А у такій галузі як медицина ціна помилки може бути занадто високою. Кожний рік через лікарські помилки частина людей отримує проблеми зі здоров’ям. Через це існує серйозна потреба зменшити вірогідність такої помилки.

Автоматизована система допомоги в діагностуванні може суттєво полегшити роботу лікарів, а також зменшити кількість лікарських помилок. З її допомогою лікарі зможуть зменшити об’єм паперової роботи, а також отримувати інформацію про вірогідний діагноз, який визначить аналітична система. Це дозволить лікарям швидше та точніше ставити діагнози.

Найбільш зручно використовувати таку систему буде в поліклініках. Терапевт, який там працює має постійний потік пацієнтів та багато паперової роботи. Тому в поліклініках дана система буде дуже корисною.

Поліклініка є медичним закладом, який надає нестаціонарну кваліфіковану і спеціалізовану медичну допомогу. В поліклініках є палати або кабінети, в яких сидять лікарі. Лікарі приймають пацієнта, який сам до них приходить. Вислуховують скарги, симптоми. Далі визначають діагноз та назначають пацієнту певне лікування. В основному, це певні ліки або медикаменти. Але можуть бути також і процедури, як, наприклад, масаж. Далі всю цю інформацію лікарі записують у особисту картку пацієнта. Наразі вони роблять це вручну, що забирає час. Для оптимізації даного процесу пропонується вести облік в електронному вигляді.

Використання автоматизованої системи допомоги в діагностуванні дозволить автоматизувати багато процесів, що дозволить пришвидшити та спростити роботу терапевта. В тому числі такі процеси як:

– Реєстрація пацієнта.

– Реєстрація візиту.

– Запис симптомів, медикаментів, процедур, схем лікування, призначених пацієнту.

– Отримання статистики про візити, медикаменти, процедури, схеми лікувань, симптоми.

– Отримання допомоги в діагностуванні хвороби на базі вказаних симптомів.

– Облік інформації про лікарів та клініки.

Отже, загалом ця система суттєво збільшить ефективність роботи лікаря, що дозволить якісніше та швидше приймати пацієнтів.

Розділ 1
Характеристика предметної області та постановка задачі

1.1 Аналіз предметної області

Поліклініка [1] є установою, яка надає медичні послуги людям. Лікарі [2], приймаючи пацієнтів [3], вислуховують скарги пацієнта, проводять обстеження та назначають пацієнту діагноз [4], певні медикаменти [5], процедури [6] та схеми лікувань [7]. Крім цього, лікарі можуть давати певні поради щодо профілактики захворювань.

Поліклініки поділяються на різні підрозділи. Кількість підрозділів залежить від розміру та типу поліклініки. Реєстратура [8] відповідає за медичні картки пацієнтів та може записати пацієнта до певного лікаря. Крім цього реєстратура реєструє нових пацієнтів. Зберігання медичних карток пацієнтів є дуже важливим, так як там зберігаються дані з попередніх візитів до лікаря, що може допомогти в діагностуванні та призначенні лікування. В деяких поліклініках є лабораторії [9], де проводять аналіз. Наприклад: загальний аналіз крові, аналіз сечі тощо.

Актуальним напрямком використання інформаційних технологій у даній області може бути автоматизація роботи поліклініки. А саме автоматизація роботи реєстратури, терапевта, лабораторії та їх взаємодії.

Лікар призначає, в основному, діагноз, симптоми, медикаменти, іноді певні процедури та схеми лікувань.

Процедурами вважаються, наприклад: ін’єкція, інгаляція, компрес, перев’язка, припікання тощо. Кожна процедура характеризується назвою, датою та деталями проведення.

Діагноз є констатацією хвороби, яка була проведена лікарем. Хвороба, або захворювання є порушенням нормальної життєдіяльності організму, внаслідок дії на нього негативних факторів. Процес, метою якого є полегшення, чи усунення симптомів і проявів захворювання, нормалізація порушених

процесів життєдіяльності називається терапією [10]. Кожний діагноз характеризується назвою, описом та симптомами, які на цей діагноз вказують.

Симптом [11] є однією із ознак хвороби, який характеризується назвою та певними деталями про обставини наявності певного симптому.

Пацієнтом вважається людина, яка отримує кваліфіковану медичну допомогу. Цю допомогу надає лікар, який є фахівцем з вищою медичною освітою.

Під час прийому пацієнта лікар збирає анамнез [12], який є сукупністю відомостей, отриманих шляхом опитування хворого або пов’язаних з ним людей. У ході спілкування з пацієнтом можна дізнатися основну інформацію про захворювання, його індивідуальні прояви. Збирання анамнезу є важливою частиною процесу діагностики. Хоча потрібно розуміти, що лише на основі анамнезу не можна встановлювати діагноз, навіть попередній. Необхідно для початку використати відомі методи дослідження хворого.

Після визначення діагнозу лікар призначає пацієнту певні медикаменти, які призначення для усунення проявів або лікування хвороби. Медикамент характеризується назвою та деталями його прийому. Такими як: скільки разів на день його приймати, в якому вигляді, до чи після їжі, тощо.

Отже, в поліклініці автоматизація дозволить сильно полегшити та оптимізувати її роботу. Автоматизований процес зберігання даних пацієнта, даних про прийом, симптоми, які були записані, медикаменти та лікування, що виписується пацієнту, статистика, а також допомога в діагностуванні допоможуть підвищити ефективність роботу терапевта та реєстратури.

1.2 Аналіз існуючого програмного забезпечення предметної області

Поліклініки використовують різноманітне програмне забезпечення для автоматизації своїх процесів. Але наразі таких систем не так багато, та жодна з них не забезпечує всі потреби, в кожної свої недоліки та переваги.

На Рисунку 1.1 зображена головна сторінка системи «medics.com» [13], яка пропонує можливості запису на прийом до лікаря, заключення договору з лікарем, перегляду графіку лікарів у всіх сімейних амбулаторіях України. Також на цьому сайті можна побачити відгуки про лікарів, записатись у електронну чергу тощо.

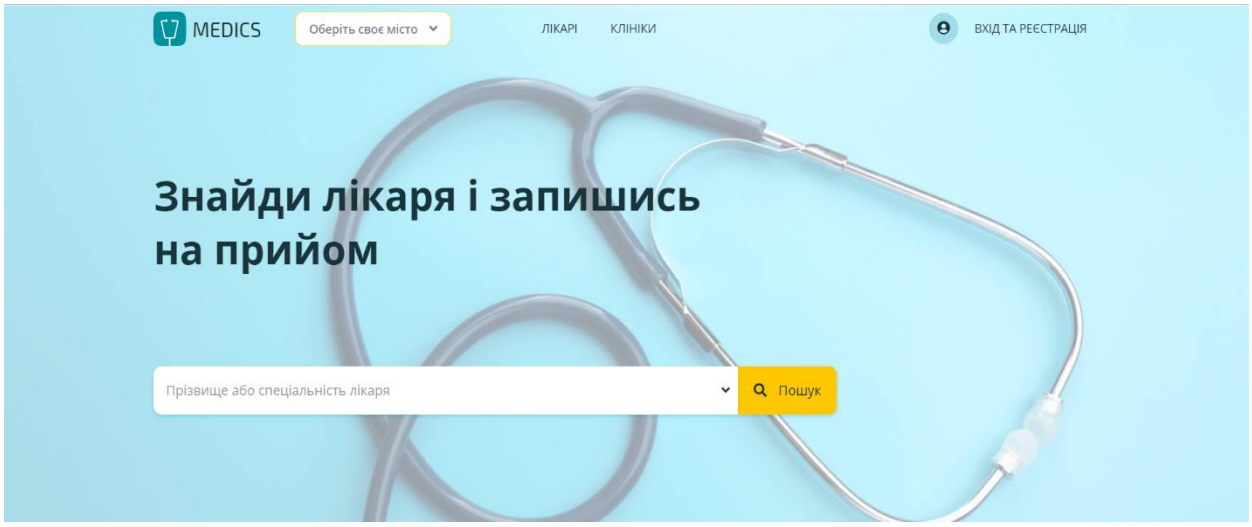


Рисунок 1.1 – Головна сторінка системи «medics.com» [13]

Дана система наразі використовується в державних поліклініках для обліку даних про пацієнтів, видання електронних направлень, запису на прийом, написання відгуків про лікарів. Недоліками цієї системи є відсутність допомоги в діагностуванні та недосконаліть системи обліку пацієнтів, яка полягає в тому, що при записі на прийом записуються лише певні особисті дані пацієнта, час та лікар, який приймає пацієнта. Але не записуються дані самого прийому, такі як: призначені симптоми, діагнози, медикаменти тощо.

На Рисунку 1.2 зображена головна сторінка системи «dpatient.bravosoft.org» [14], яка дозволяє пацієнту зберігати його особисті, фінансові, медичні дані. Крім цього дана система дозволяє зберігати дані лікарів, автоматизувати роботу реєстратури, лабораторії, полегшує ведення медичної документації. Дана система дозволяє автоматизувати роботу поліклініки та лікарні, але основним недоліком є недостатньо інтуїтивно зрозумілий інтерфейс користувача.

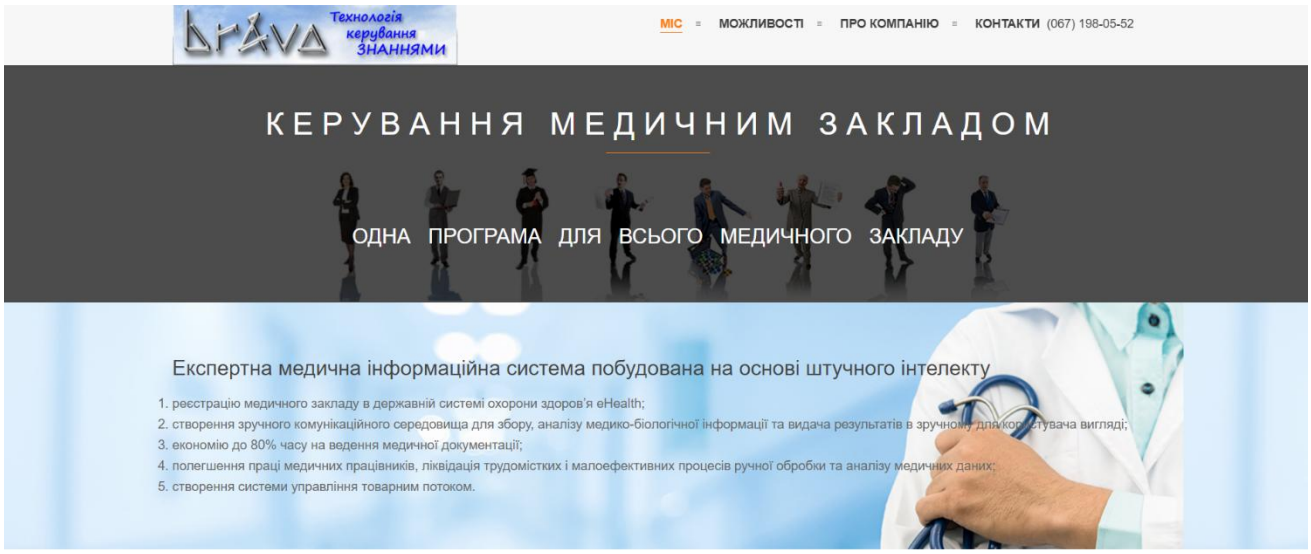


Рисунок 1.2 – Головна сторінка системи «dpatient.bravosoft.org» [14]

Також є система «Medcard24» [15]. Вона дозволяє пацієнту мати свій власний особистий кабінет, в якому він може записатись до лікаря, подивитись назначений лікарем план лікування, подивитись свою медичну картку. Крім цього дана система дозволяє заводити картки не тільки для конкретного пацієнта, а і для його сім'ї. На рисунку 1.3 зображена сторінка з медичними картками цієї системи.

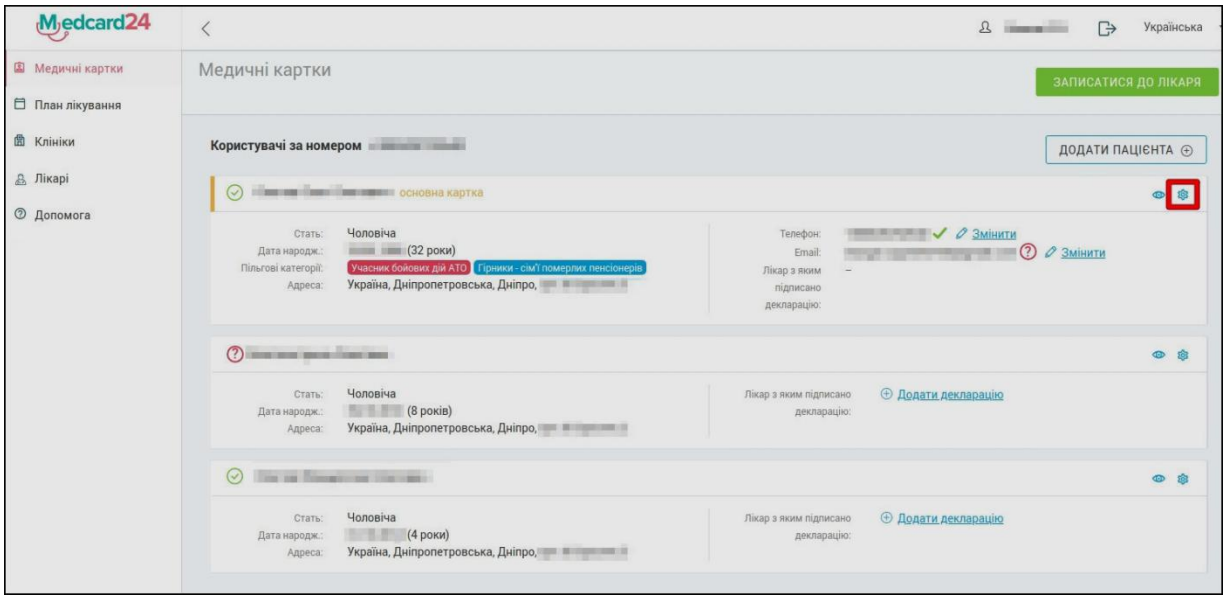


Рисунок 1.3 – Сторінка з медичними картками системи «Medcard24» [15]

Загалом, дана система відповідає багатьом вимогам пацієнтів та лікарів, але в ній відсутня система допомоги в діагностуванні хвороби терапевтом, статистика відвідувань та можливість лікарю додавати нові симптоми, схеми лікувань, хвороби, процедури у систему.

Отже, наразі можна знайти багато прикладів автоматизованих інформаційних систем поліклініки. В основному, вони надають функціональні можливості пацієнтам записатись на прийом, підписати договір з лікарем та робити інші речі, зв’язані зі звичайним прийомом у лікаря, але є і ті, які дозволяють отримувати діагностику, вести облік пацієнтів, автоматизувати ведення медичної документації. Вони всі мають свої переваги та недоліки, але загалом, напрямок автоматизації роботи терапевтів у поліклініках є актуальним.

1.3 Аналіз сучасних засобів створення програмного забезпечення

Для створення додатку можна вибрати веб-застосування [16], віконне десктоп-застосування [17], чи застосування для мобільних пристроїв [18].

Веб-застосування використовується загалом для додатків, в яких потрібне постійне інтернет-з’єднання. Для взаємодії з іншими користувачами чи для доступу до бази даних. Основною перевагою даного типу застосування є те, що його не потрібно встановлювати, для його роботи необхідно лише мати веб-браузер [19]. Основним недоліком є постійна необхідність в інтернет-з’єднанні.

Віконне десктоп-застосування використовується, в основному, в тих випадках коли не потрібне інтернет-з’єднання. Основною перевагою даного типу застосування є те, що часто воно дає більше можливостей користувачу та більшу гарантію безпеки. Основним недоліком є те, що такий тип застосуванням необхідно встановлювати.

Застосування для мобільних пристроїв. загалом, дуже схоже на віконне-десктоп застосування, але використовується воно для мобільних пристроїв, що є як перевагою, так і недоліком такого типу застосувань.

На даний момент можна вибрати із чотирьох найбільш популярних платформ для створення застосувань: .NET [20], Java [21], PHP [22], JavaScript [23].

Microsoft .NET – програмна технологія від компанії Microsoft, яка діє як платформа для створення звичайних програм, так і веб-застосунків. Це крос-платформна технологія. Вона складається з двох основних частин – середовище виконання та інструментарій розробки. На даній платформі можна створювати десктоп-застосунки за допомогою таких мов як: C++[24], C#[25], Visual Basic[26], а також використовувати такі СКБД як MS SQL Server[27]. Також до цієї платформи належить технологія ASP.NET[28], яку можна використовувати для створення веб-застосунків на мові C#. Загалом, дана платформа є достатньо хорошою для створення надійних веб-додатків, але основним недоліком є занадто сильна прив'язаність до ОС Windows[29] та продуктів Microsoft[30] загалом. Також недоліком є те, що в середовищі Microsoft Visual Studio[31] недостатньо зручно реалізовувати front-end[32] за допомогою різноманітних фреймворків[33], типу React[34] чи Angular[35], порівняно з платформою JavaScript.

PHP (Hypertext Preprocessor – гіпертекстовий препроцесор) – скриптова мова програмування, яка призначена для генерації HTML[36]-сторінок на стороні веб-сервера. PHP є дуже поширеною мовою програмування, яка використовується для серверної частини додатку. Код PHP інтерпретується у HTML-код, а потім передається на сторону клієнта. Порівняно з JavaScript, це краще з точки зору безпеки, але водночас це погіршує інтерактивність сторінок. Full-Stack додаток із використання JavaScript для front-end та PHP для back-end може вирішити цю проблему. Багато сучасних додатків зроблені саме таким чином. Сервер, написаний на PHP передає дані у форматі JSON[37], які далі оброблює клієнт за допомогою JavaScript. Дана платформа підходить як під ОС Linux[38], так і під ОС Windows. Крім цього платформа PHP може використовувати різні СКБД, включаючи MySQL, PostgreSQL[39], Microsoft SQL і т.д., що є перевагою порівняно з платформою .Net. Недоліком даної

платформи є недостатня, порівняно з JavaScript платформою зручність роботи в Full-Stack додатках.

JavaScript (JS) – скриптова мова програмування з динамічною типізацією. В цих аспектах вона схожа на PHP, але відрізняється від C#. JavaScript поєднує в собі декілька парадигм, таких як: об’єктно-орієнтоване програмування та функціональне програмування. Найчастіше, дана мова програмування використовується для клієнтської частини додатку, для взаємодії з користувачем, динамічної зміни контенту веб-сторінки, створення інтерактивності. Але крім цього, JavaScript також можна використовувати і для створення серверної частини. За допомогою фреймворків, доступних даній платформі, це робити доволі зручно. В основному такі фреймворки не спонукають користувача використовувати шаблон проєктування MVC[40], як це робить ASP.NET MVC (.NET) або Laravel (PHP), що дає більше ступінь свободи у розробці. Серверна частина, написана на JavaScript, передає дані у форматі JSON клієнтській частині схожим чином як це роблять .NET чи PHP фреймворки. Але загалом, це робити зручніше на платформі JavaScript через близкість до неї формату JSON та завдяки тому, що клієнтська та серверна частини використовують одну мову програмування. Також потрібно розуміти, що платформа JavaScript є найбільш багато-платформною, так як з її допомогою можна писати не тільки серверну та клієнтську частини веб-додатку, а також мобільні додатки, додатки для персональних комп’ютерів.

Усі чотири платформи мають свої переваги та недоліки але загалом, але так як за допомогою JavaScript можна легко і зручно створювати як клієнтську, так і серверну частину додатку, а також через велику кількість доступних фреймворків, було обрано платформу JavaScript.

1.4 Постановка задачі та вимоги до розробки інформаційної системи

Метою кваліфікаційної роботи бакалавра є розробка автоматизованої системи допомоги в діагностуванні хвороб терапевтом на платформі JavaScript, що виконує наступні основні функції:

1. Робота з відкритою інформацією (виведення інформації про лікарів та поліклініки, що зареєстровані в системі).
2. Допомога в діагностуванні хвороби.
3. Робота з візитами (додавання симптомів пацієнта, діагнозів, направлень на процедури, рецептів на медикаменти).
4. Робота з картками пацієнтів (додавання симптомів пацієнта, діагнозів, направлень на процедури, рецептів на медикаменти, редагування його особистої інформації).
5. Робота з особистими даними (реєстрація в системі, авторизація, редагування даних профілю).
6. Робота з внутрішніми даними системи (хвороби, симптоми, медикаменти, процедури, працівники, пацієнти, палати).
7. Робота зі статистикою роботи терапевта (по симптомах пацієнта, діагнозах, направленнях на процедури, рецептах на медикаменти).

Розділ 2

Проєктування інформаційної системи

2.1 Функціональна структура та бізнес-процеси системи

При розробці автоматизованої системи допомоги в діагностуванні хвороби терапевтом необхідно автоматизувати наступні бізнес-процеси:

1. Бізнес-процес «Робота з відкритою інформацією».
2. Бізнес-процес «Робота з візитами».
3. Бізнес-процес «Робота з картками пацієнтів».
4. Бізнес-процес «Робота з особистими даними».
5. Бізнес-процес «Робота з внутрішніми даними системи».
6. Бізнес-процес «Робота зі статистикою роботи терапевта».
7. Бізнес-процес «Робота з системою допомоги в діагностуванні».

Бізнес-процес «Робота з відкритою інформацією». Система повинна надавати доступ неавторизованим користувачам до відкритої інформації, такої як:

- Відомості про медичні заклади, а саме їх адреса, розклад, профіль і т.д.
- Відомості про лікарів, їх спеціалізація, часи прийому тощо.

Доступ користувача до цієї інформації відбувається наступним чином:

1. Користувач відкриває сторінку «Клініки» та бачить там всю відкриту інформацію про медичні заклади, присутні у системі.
2. Користувач відкриває сторінку «Лікарі» та бачить всю відкриту інформацію про лікарів, присутніх у системі.

Діаграму дій для даного бізнес-процесу зображено на рисунку 2.1.

Бізнес-процес «Робота з візитами». Користувач повинен мати можливість працювати з візитами до нього пацієнтів. Він повинен мати можливість:

- Додавання нових візитів.
- Бачити інформацію про старі візити.
- Здійснювати пошук візитів.

– Записувати симптоми, медикаменти, процедури, схеми лікувань, діагнози до певного візиту.

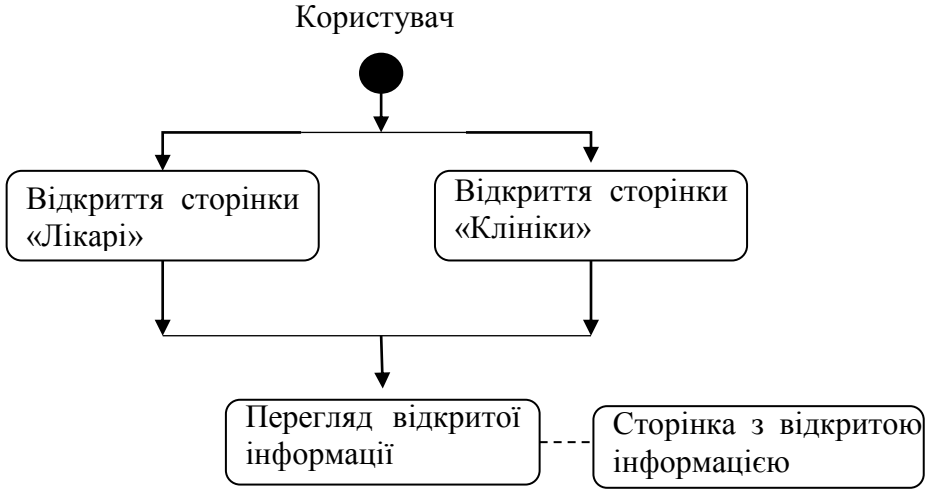


Рисунок 2.1 – Діаграма дій бізнес-процесу «Робота з відкритою інформацією»

Робота користувача з візитами відбувається наступним чином:

1. Користувач реєструється та авторизується у системі.

2. Відкриває сторінку «Візити».

3. Використовуючи інтерфейс сторінки, здійснює пошук, переходить до картки пацієнта та додає нові візити.

Діаграма дій даного бізнес-процесу зображена на рисунку 2.2.

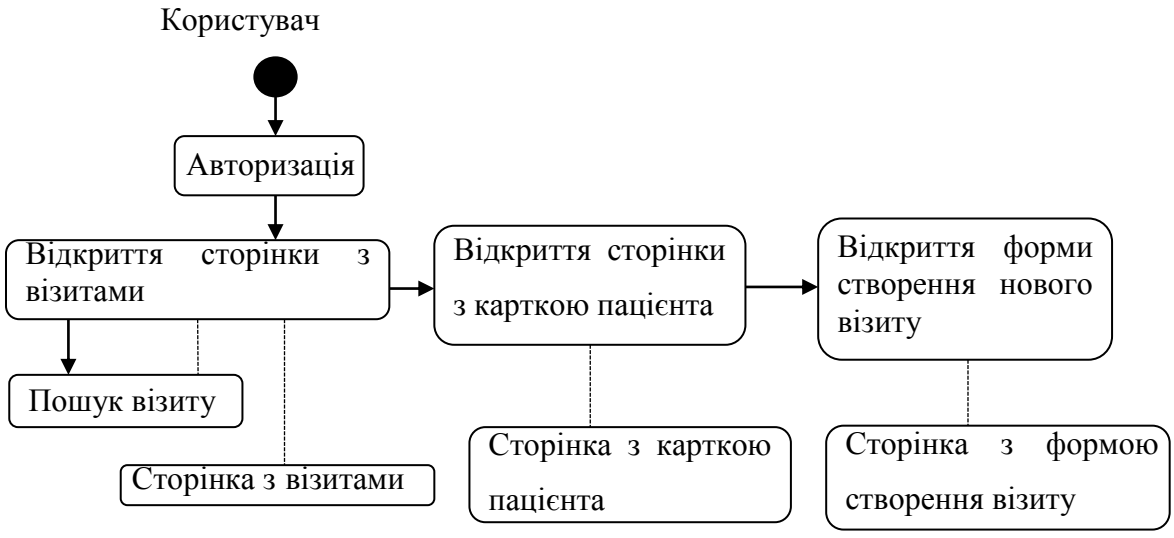


Рисунок 2.2 – Діаграма дій бізнес-процесу «Робота з візитами»

Бізнес-процес «Робота з картками пацієнтів». Система повинна надавати можливість користувачу працювати з особистими даними пацієнтів, з їх електронними картками. А саме користувач має мати можливість:

– Передивлятись, редагувати, додавати, видаляти особисті дані пацієнтів.

– Мати доступ до інформації про симптоми, медикаменти, процедури, схеми лікувань, діагнози призначені пацієнту на попередніх візитах.

Робота з картками пацієнта здійснюється наступним чином:

1. Користувач відкриває сторінку «Пацієнти».

2. За допомогою інтерфейсу сторінки передивляється, додає, редагує та видаляє дані про пацієнтів.

Діаграма дій даного бізнес-процесу зображена на рисунку 2.3.

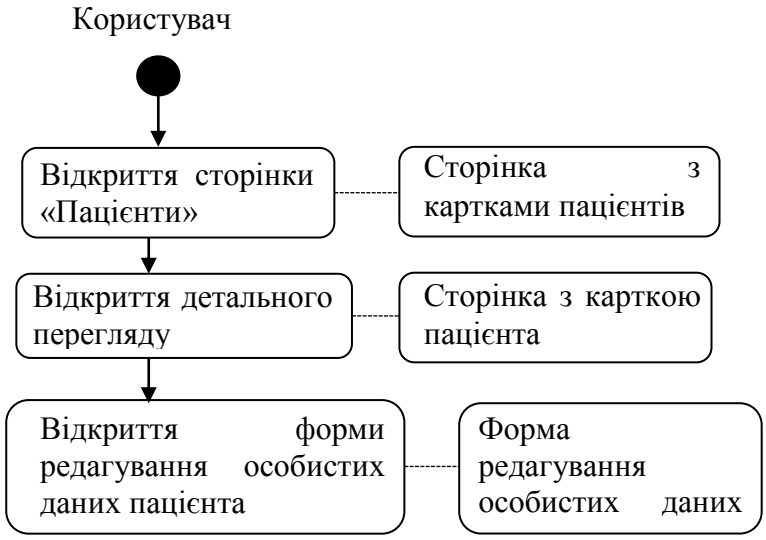


Рисунок 2.3 – Діаграма дій бізнес-процесу «Робота з картками пацієнтів»

Бізнес-процес «Робота з особистими даними». Система повинна надавати користувачу можливість реєстраціє, авторизації та, загалом, працювати з його особистими даними. Робота з особистими даними користувача відбувається наступним чином.

1. Користувач відкриває сторінку «Реєстрація».

2. Використовуючи інтерфейс сторінки, реєструється.

3. Відкриває сторінку «Авторизація».

4. Використовуючи інтерфейс сторінки, авторизується.

5. Відкриває сторінку «Особисті дані».

6. Використовуючи інтерфейс сторінки, переглядає, додає, редагує та видаляє свої особисті дані.

Діаграма дій для даного бізнес-процесу зображена на рисунку 2.4.

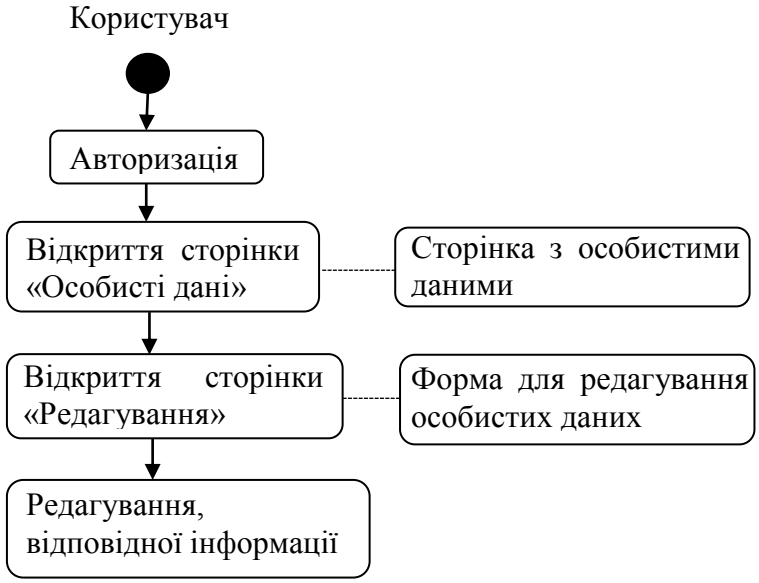


Рисунок 2.4 – Діаграма дій для бізнес-процесу «Робота з особистими даними»

Бізнес-процес «Робота з внутрішніми даними системи». Система повинна надавати користувачу можливість працювати з внутрішніми даними системи, такими як: симптоми, медикаменти, виробники тощо. Робота з внутрішніми даними системи відбувається наступним чином:

1. Користувач відкриває сторінку «Внутрішні дані».

2. Використовуючи інтерфейс сторінки, переглядає, додає, видаляє, редагує внутрішні дані системи.

Діаграма дій для даного бізнес-процесу зображена на рисунку 2.5.

Бізнес-процес «Робота зі статистикою роботи терапевта». Система повинна давати можливість користувачу бачити статистику свої роботи у вигляді діаграм, таблиць, графіків. Користувач повинен мати статистику по візитам, пацієнтам, симптомам, хворобам, процедурам, схемам лікувань. Робота зі статистикою здійснюється наступним чином:

1. Користувач відкриває сторінку «Статистика».

2. Використовуючи інтерфейс сторінки, переглядає статистику своєї роботи у вигляді графіків, діаграм, таблиць.

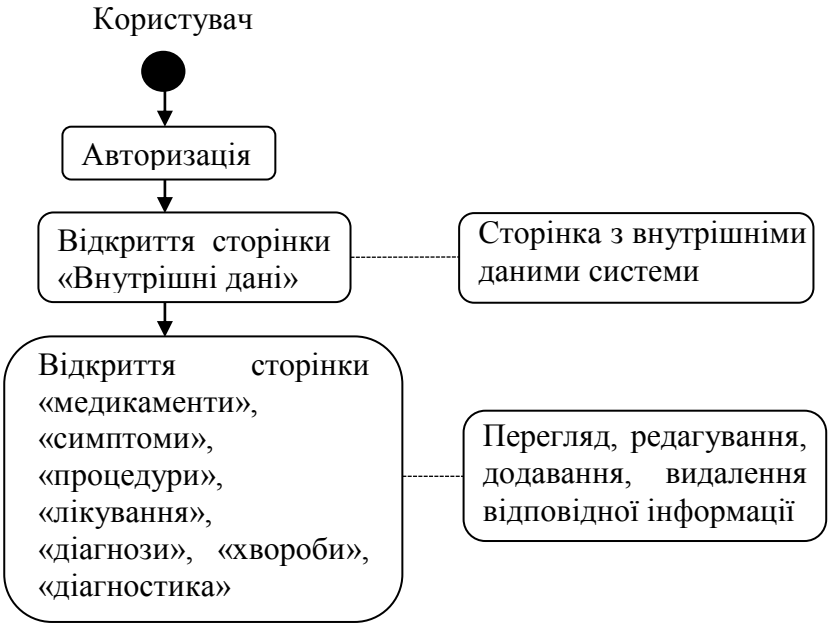


Рисунок 2.5 – Діаграма дій для бізнес-процесу «Робота з внутрішніми даними системи»

Діаграма дій для даного бізнес-процесу зображена на рисунку 2.6.

Рисунок 2.6 – Діаграма дій для бізнес-процесу «Робота зі статистикою роботи терапевта»

Бізнес-процес «Робота з системою допомоги в діагностуванні». Система повинна давати можливість користувачу отримувати певні поради щодо діагнозу пацієнта. Система аналізуватиме статистику діагнозів, хвороб, симптомів та буде давати приблизний діагноз. Робота з системою допомоги в діагностуванні відбувається наступним чином:

1. Користувач реєструється та авторизується у системі.
2. Заходить на сторінку «Діагностика».
3. Вводить симптоми.
4. Натискає кнопку «Отримати приблизний діагноз».
5. Система, проаналізувавши дані, які в ній вже є, видає приблизні можливі діагнози пацієнта.

Діаграма дій для даного бізнес-процесу зображена на рисунку 2.7.

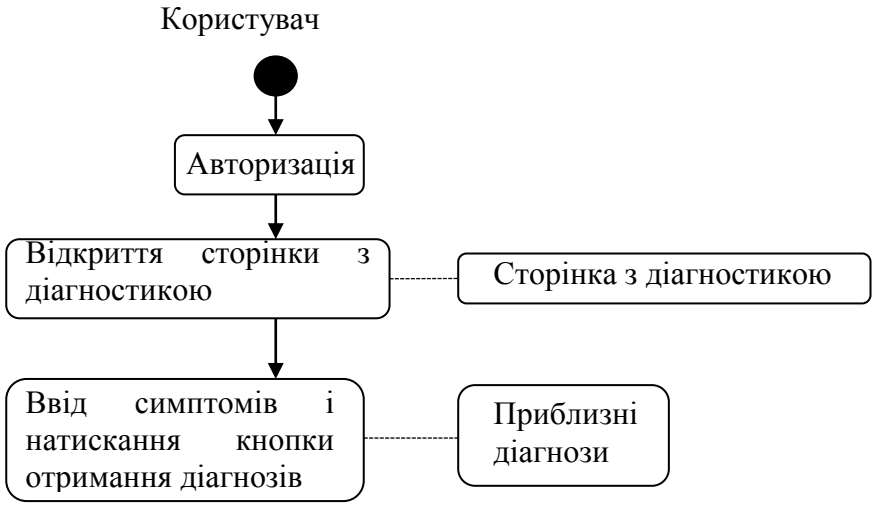


Рисунок 2.7 – Діаграма дій для бізнес-процесу «Робота з системою допомоги в діагностуванні»

Отже, система повинна мати зазначену функціональну структуру та реалізувати відповідні бізнес-процеси.

2.2 Інформаційна структура системи

Інформаційна структура системи поблована на тому, що дані будуть передаватись із бази даних на сервер, з сервера в проміжне сховище, а звідти клієнту. Відповідну схему інформаційної структури зображено на рисунку 2.8.

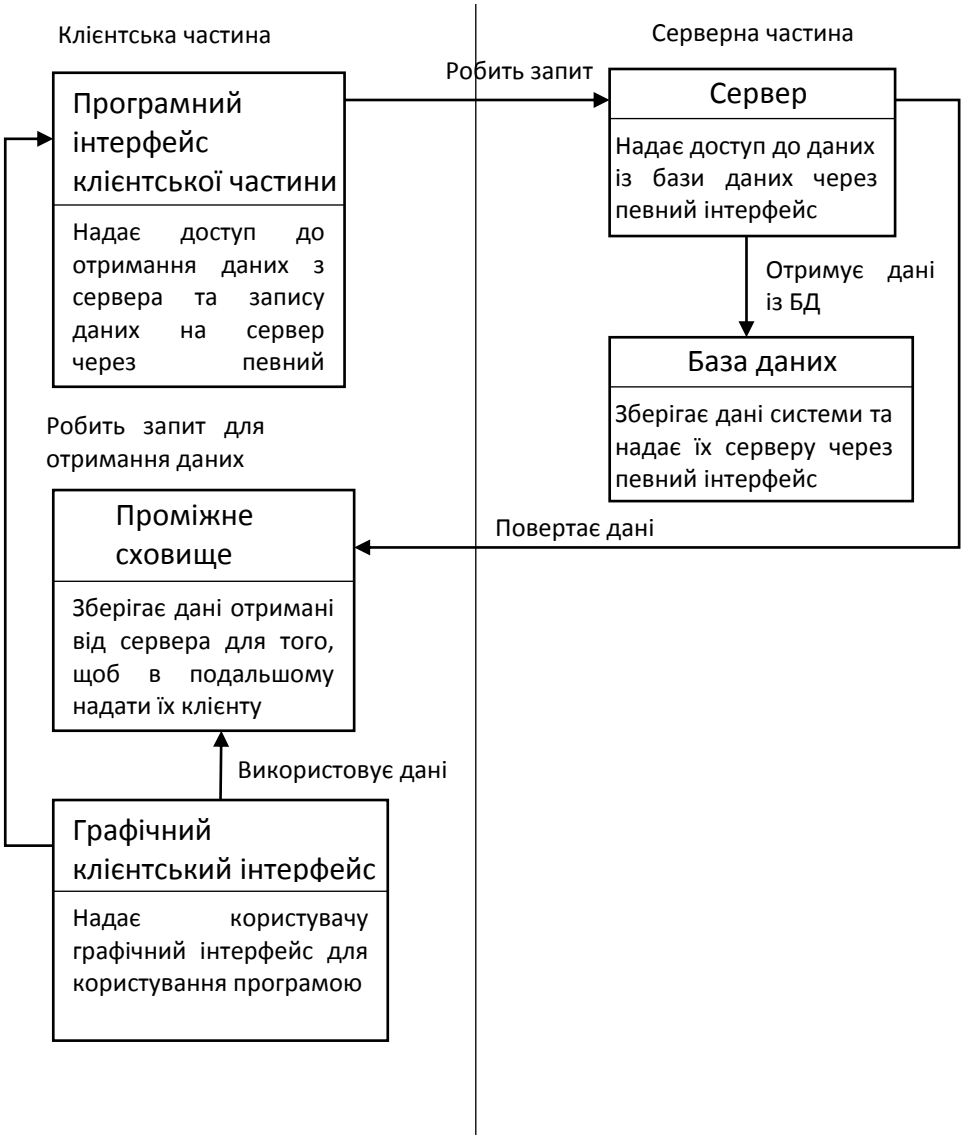


Рисунок 2.8 – Схема інформаційної структури системи

Як видно з рисунку 2.8, клієнтська частина надає певний інтерфейс, за допомогою якого можна звертатися до сервера для отримання чи запису даних. Далі сервер звертається до бази даних, яка знаходиться у серверній частині системи, для отримання чи запису певних даних. Після отримання сервером

певних даних, вони записуються у проміжне сховище, яке знаходиться у клієнтській частині системи та звідки клієнт може отримати необхідні йому дані.

Для забезпечення збереження даних було розроблено БД, даталогічна модель якої зображена на рисунку 2.9.

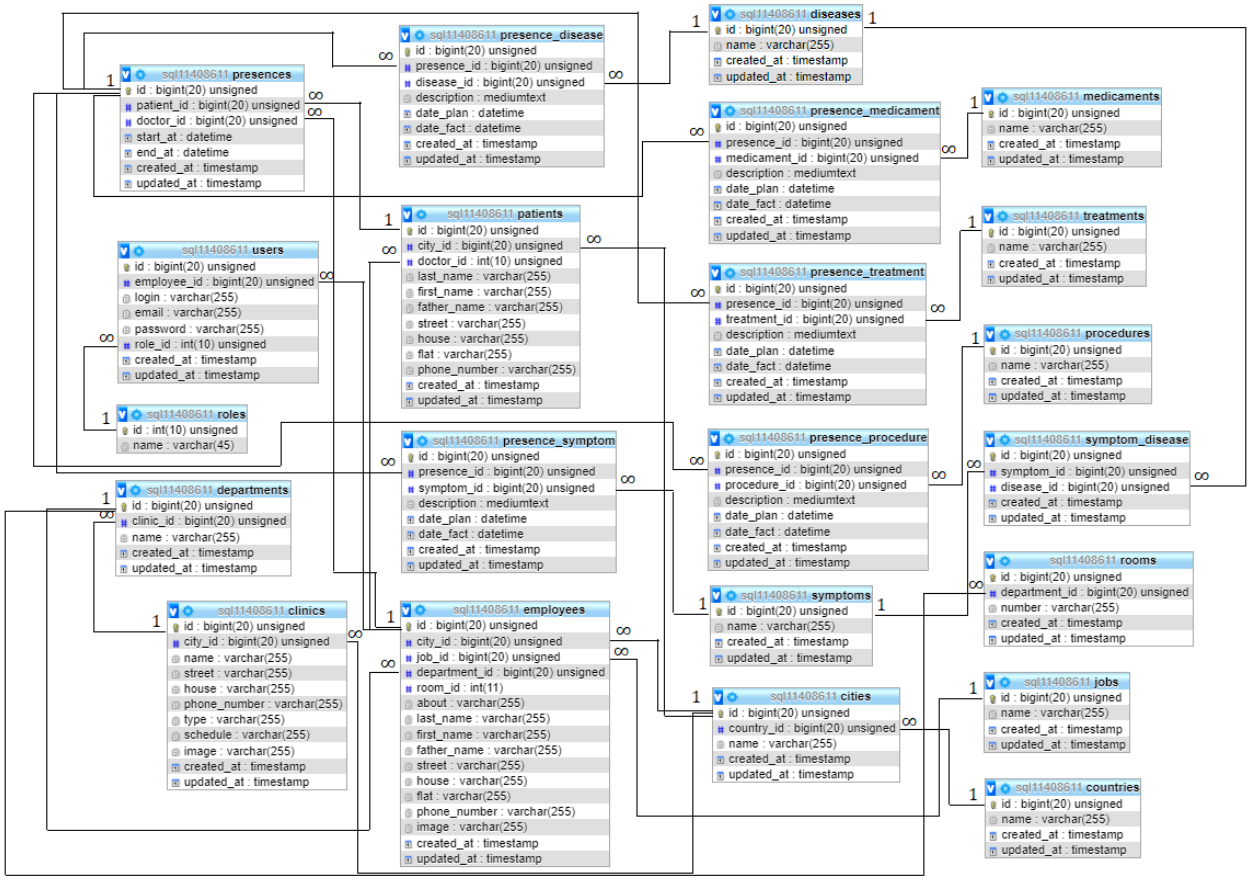


Рисунок 2.9 – Схема бази даних

Дана база даних знаходиться у 3 Нормальній формі. Головною таблицею є таблиця «Presences», у яку заносяться пацієнт, призначений лікар, дата початку та закінчення візиту.

Пацієнти представлені таблицею «Patients», в яку заноситься прізвище, ім’я, по-батькові, номер телефона та адреса пацієнта.

Працівники поліклініки представлені таблицею «Employees», в яку заноситься прізвище, ім’я, по-батькові, номер телефона, адреса, фотографія, посада та відділ працівника. Додатковою таблицею для користувачів є таблиця «Users», куди заносяться дані для авторизації, такі як: логін, пароль, адреса електронної пошти, роль.

Посади заносяться у таблицю «Jobs». Симптоми заносяться у таблицю «Symptoms». Процедури у таблицю «Procedures». Схеми лікувань у таблицю «Treatment». Хвороби у таблицю «Diseases». Ролі у таблицю «Roles». Країни у таблицю «Countries». Процедури, медикаменти, симптоми, хвороби, схеми лікувань, ролі, посади мають лише поле назви.

Палати заносяться у таблицю «Rooms». В цю таблицю заносяться: номер палати та відділ, до якого вона відноситься.

Відділи заносяться у таблицю «Departments». В цю таблицю заносяться: назва та клініка, до якої належить клініка.

Клініки заносяться у таблицю «Clinics». В цю таблицю заносяться: назва, місто, адреса, графік роботи, тип, номер телефону, фотографія.

Міста заносяться у таблицю «Cities». В цю таблицю заносяться: назва, країна до якої належить місто.

Симптоми, медикаменти, діагнози, процедури, схеми лікувань заносяться у відповідні таблиці, які є розвідними між таблицями «Symptoms», «Diseases», «Procedures», «Medicaments», «Treatments» та таблицею «Presences» і називаються відповідно «Presence_Symptom», «Presence_Disease», «Presence_Procedure», «Presence_Medicament», «Presence_Treatment». В дані таблиці заноситься id візиту, id сутності, що призначається під час цього візиту, і доповнення щодо призначення.

Дані для системи діагностики записуються у розвідну таблицю «Symptom_Disease». В неї записується: id симптому та id хвороби.

Таблиця «Presence» призначена для занесення туди даних про візити. Вона задіяна для багатьох бізнес-процесів, по суті, всіх бізнес-процесів, пов’язаних з візитами: робота з даними візиту, пацієнтом, його симптомами, діагнозами і т.д. Основна взаємодія з цією таблицею відбувається через розвідні таблиці з таблицями, які пов’язані з візитами, такими як: медикаменти, діагнози, схеми лікувань, симптоми і т.д. Наприклад: додавання, виведення, видалення, редагування візитів.

Таблиця «Patients» призначена для занесення туди даних про пацієнтів. Дана таблиця призначена для бізнес-процесів, пов’язаних з самими пацієнтами, їх особистими даними. Вона часто взаємодіє з таблицею з візитами. Наприклад: додавання, видалення, редагування, виведення даних про пацієнтів.

Таблиця «Users» призначена для занесення туди даних про користувачів. Дана таблиця призначена для роботи з бізнес-процесами, пов’язаними з користувачами системи, їх особистою інформацією, реєстрацією та авторизацією. Наприклад: реєстрація, авторизація, виведення особистих даних користувачів.

Таблиця «Employees» призначена для занесення туди даних про працівників. Дана таблиця призначена розширити таблицю «Users» даними про користувачів.

Таблиця «Cities» призначена для занесення туди даних про міста. Дана таблиця використовується в бізнес-процесах, пов’язаних з містами. Це в основному взаємодія з особистими даними пацієнтів або користувачів. Наприклад: додавання, виведення, видалення, редагування даних про міста.

Таблиця «Clinics» призначена для занесення туди даних про клініки. Дана таблиця використовується для бізнес-процесів, пов’язаних з клініками. В основному, це процеси реєстрації нового користувача чи пацієнта. Наприклад: додавання, видалення, виведення, редагування даних про клініки.

Таблиця «Countries» призначена для занесення туди даних про країни. Дана таблиця використовується в бізнес-процесах, пов’язаних з країнами. Це в основному взаємодія з особистими даними пацієнтів або користувачів. Наприклад: додавання, видалення, виведення, редагування даних про країни.

Таблиця «Departments» призначена для занесення туди даних про відділи. Дана таблиця використовується в бізнес-процесах, пов’язаних з різними відділами поліклінік. Це в основному взаємодія з особистими даними пацієнтів або користувачів. Наприклад: додавання, видалення, виведення, редагування даних про відділи.

Таблиця «Diseases» призначена для занесення туди даних про хвороби. Дана таблиця працює з бізнес-процесами, пов’язаними з хворобами. Це, в основному, реєстрація нових хвороб у системі, або записування певних хвороб пацієнту під час візиту. Наприклад: додавання, видалення, виведення, редагування даних про хвороби.

Таблиця «Jobs» призначена для занесення туди даних про посади. Дана таблиця призначена для роботи з бізнес-процесами, пов’язаними з особистими даними користувачів, а саме з їхніми посадами. Наприклад: додавання, видалення, виведення, редагування даних про посади.

Таблиця «Medicaments» призначена для занесення туди даних про медикаменти. Дана таблиця задіяна в бізнес-процесами, пов’язаними з візитом пацієнта та призначенням йому певних ліків. Наприклад: додавання, видалення, виведення, редагування даних про медикаменти.

Таблиця «Procedures» призначена для занесення туди даних про процедури. Дана таблиця задіяна в бізнес-процесами, пов’язаними з візитом пацієнта та призначенням йому певних процедур. Наприклад: додавання, видалення, виведення, редагування даних про процедури.

Таблиця «Rooms» призначена для занесення туди даних про палати. Дана таблиця задіяна в бізнес-процесами, пов’язаними з візитом пацієнта та кабінетом, який він відвідує. Наприклад: додавання, видалення, виведення, редагування даних про кабінети.

Таблиця «Symptoms» призначена для занесення туди даних про симптоми. Дана таблиця задіяна в бізнес-процесами, пов’язаними з візитом пацієнта та призначенням йому певних симптомів. Наприклад: додавання, видалення, виведення, редагування даних про симптоми.

Таблиця «Treatments» призначена для занесення туди даних про схеми лікувань. Дана таблиця задіяна в бізнес-процесами, пов’язаними з візитом пацієнта та призначенням йому певних схем лікувань. Наприклад: додавання, видалення, виведення, редагування даних про схеми лікувань.

Таблиця «Presence_Disease» призначена для забезпечення роботи аналітичної системи, яка буде допомагати терапевту в діагностуванню. Наприклад: запису історії хвороби пацієнта, виведення порад щодо діагнозу.

Таблиця «Presence_Procedure» призначена для зберігання історії процедур певного пацієнта. Наприклад: додавання, видалення, редагування, виведення процедур, призначених пацієнту під час візиту.

Таблиця «Presence_Symptom» призначена забезпечити запис історії симптомів певного пацієнта під час певного візиту. Наприклад: додавання, видалення, редагування, виведення процедур, призначених пацієнту під час візиту.

Таблиця «Presence_Treatment» призначена забезпечити запис історії схем лікувань, які були виписані певному пацієнту під час візиту. Наприклад: додавання, видалення, редагування, виведення схем лікування, призначених пацієнту під час візиту.

Таблиця «Symptom_Disease» призначена зв’язати симптоми та хвороби, щоб використовувати це для аналізу та видання приблизного діагнозу.

Таблиці «Presences», «Presence_Symptom», «Presence_Medicament», «Presence_Procedure», «Presence_Disease», «Presence_Treatment», «Medicaments», «Symptoms», «Procedures», «Treatments», «Diseases» використовуються для бізнес-процесу «Робота з візитами». Користувач за допомогою цих таблиць додає, видаляє, редагує нові візити. А саме, він записує нові симптоми, схеми лікувань, медикаменти, діагнози, процедури певному пацієнту, який прийшов на візит.

Таблиці «Presences», «Presence_Symptom», «Presence_Medicament», «Presence_Procedure», «Presence_Disease», «Presence_Treatment», «Medicaments», «Symptoms», «Procedures», «Treatments», «Diseases», «Rooms», «Patients» використовуються у бізнес-процесі «Робота з картками пацієнта». Використовуючи ці таблиці, користувач може переглядати, історію візитів пацієнта, його попередні симптоми, хвороби, схеми лікувань, медикаменти, процедури, які йому виписував лікар.

Таблиці «Symptom_Disease», «Symptoms», «Diseases», «Presence_Symptom» використовуються для бізнес-процесу «Робота з системою допомоги в діагностуванні». Користувач, встановлюючи діагноз пацієнту, може скористатись цією системою для отримання приблизного діагнозу, який буде отриманий, аналізуючи записи в даних таблицях.

Таблиці «Rooms», «Users», «Employees», «Jobs», «Cities», «Countries», «Departments», «Clinics» використовуються у бізнес-процесі «Робота з особистими даними». Користувач за допомогою цих таблиць зберігає та редагує свої особисті дані.

Таблиці «Presences», «Presence_Symptom», «Presence_Medicament», «Presence_Procedure», «Presence_Disease», «Presence_Treatment», «Medicaments», «Symptoms», «Procedures», «Treatments», «Diseases», «Rooms», «Users», «Employees», «Jobs», «Cities», «Countries», «Departments», «Clinics» використовуються для бізнес-процесу «Робота зі статистикою роботи терапевта». Система, використовуючи ці таблиці, надає користувачу статистику відносно його роботи, а також загальну статистику відносно пацієнтів.

Таблиці «Users», «Employees», «Jobs», «Cities», «Countries», «Departments», «Clinics» використовуються для бізнес-процесу «Робота з відкритими даними». За допомогою цих таблиць, система видає користувачу інформацію про клініки та лікарів, які йому доступні.

Таблиці «Medicaments», «Symptoms», «Procedures», «Treatments», «Diseases», «Rooms» використовуються для бізнес-процесу «Робота з внутрішніми даними системи». Система, використовуючи ці таблиці, надає користувачу доступ до інформації про медикаменти, хвороби, процедури, схеми лікувань, хвороби, палати, які були додані не тільки ним, а й іншими користувачами.

Отже, сформований обсяг таблиць є достатнім для обробки всіх необхідних даних для роботи з автоматизованою системою допомоги в діагностуванні хвороб терапевтом.

2.3 Вибір засобів розробки інформаційної системи

Для розробки автоматизованої системи допомоги в діагностуванні хвороби терапевтом слід обрати мову програмування, СКБД та фреймворк.

2.3.1 Вибір мови програмування

JavaScript (JS) – скриптова мова програмування з динамічною типізацією. В цих аспектах вона схожа на PHP, але відрізняється від C#. JavaScript поєднує в собі декілька парадигм, таких як: об’єктно-орієнтоване програмування та функціональне програмування. Також в даній мові присутній автоматичний «збір сміття», тобто на відміну від C++, програмісту не потрібно думати про очищення пам’яті від раніше виділених об’єктів. Найчастіше, дана мова програмування використовується для клієнтської частини додатку, для взаємодії з користувачем, динамічної зміни контенту веб-сторінки, створення інтерактивності, взаємодії з DOM [40]. Але крім цього, JavaScript також можна використовувати і для створення серверної частини.

Працюючи з JavaScript на серверній частині додатку, ми, в основному, передаєм дані у форматі JSON на сторону клієнта. Завдяки близькості формату JSON цієї мови програмування, робити це відносно просто. Також потрібно розуміти, що за допомогою JavaScript можна писати не тільки серверну та клієнтську частини веб-додатку, а також мобільні додатки та додатки для персональних комп’ютерів.

2.3.2 Вибір фреймворку

Для створення даного проєкту було обрано 3 фреймворки. Один для клієнтської частини – React, один для серверної частини – Express[41], один для зв'язування клієнтської та серверної частини – Redux[42].

React – JavaScript-бібліотека з відкритим вихідним кодом. Вона призначена для розробки клієнтської частини додатку. React Native може використовуватись для написання мобільних додатків. Основна задумка даного фреймворку полягає в тому, що додаток ділиться на певні компоненти, що зменшує дублювання коду. У кожного компонента є свій стан, в якому можуть зберігатися дані, а також він може приймати дані від інших компонентів. Крім цього в компонента є можливість задавати поведінку при різних подіях, таких як: створення компонента, знищення компонента, зміна вхідних даних від іншого компонента і т.д. Дані клієнтська частина, яка використовує цей фреймворк отримує через REST [43] API [44] у форматі JSON. Даний фреймворк працює у клієнтській частині додатку.

Для більш зручного управління даними додатку використовується фреймворк Redux. Він дозволяє поділити роботу з даними на певні функціональні блоки, кожен з яких буде складатися з контейнера та контролера. Контейнер буде зберігати необхідні дані. А контролер посилати запити на сервер для того, щоб отримати певні дані від моделі. Даний фреймворк працює у клієнтській частині додатку.

Для роботи серверної частини було обрано фреймворк Express. Це фреймворк, який працює в середині середовища Node.js [45], тобто з його допомогою можна створювати серверну частину додатку. Даний фреймворк не заставляє програміста використовувати якийсь певний архітектурний шаблон, типу MVC. І це є однією з переваг даного фреймворку. З його допомогою ми створимо моделі, які будуть відповідати за бізнес-логіку додатку. Вони, реагуючи на певний запит від контролера, будуть звертатися до БД, брати необхідні дані та вертати їх контролеру, який в свою чергу запише їх у контейнер. Даний фреймворк працює у серверній частині додатку.

Отже, для виконання даного проєкту було обрано 3 фреймворки. Один для клієнтської частини – React, один для серверної частини – Express, один для зв'язування клієнтської та серверної частини – Redux, так як вони дозволяють створювати повноцінні Full Stack додатки та мають якісну документацію.

2.3.3 Вибір СКБД

Для виконання даного проєкту було обрано СКБД MySQL [46] – систему керування реляційними базами даних, що була створена для підвищення швидкодії обробки великих баз даних, як альтернатива комерційним системам. MySQL використовує мову запитів SQL [47], є доволі швидкодіючою та підходить як для середніх, так і для великих проєктів.

MySQL працює з багатьма фреймворками, працює на багатьох платформах, має якісну документацію, високі стандарти безпеки, велику кількість розробників, які використовують дану СКБД. Крім цього завдяки популярності даної СКБД, є висока ймовірність, що вона буде і надалі дороблятися її розробниками.

Відповідно до переваг і недоліків різноманітних СКБД було обрано MySQL, так як вона підходить під платформу JavaScript, має хорошу швидкодію, надійність та хорошу документацію.

Розділ 3

Програмна реалізація інформаційної системи

3.1 Структура та функціональне призначення складових системи

В даному проєкті кожний компонент буває 5 видів, а взаємодія між ними відбувається так як показано на рисунку 3.1.

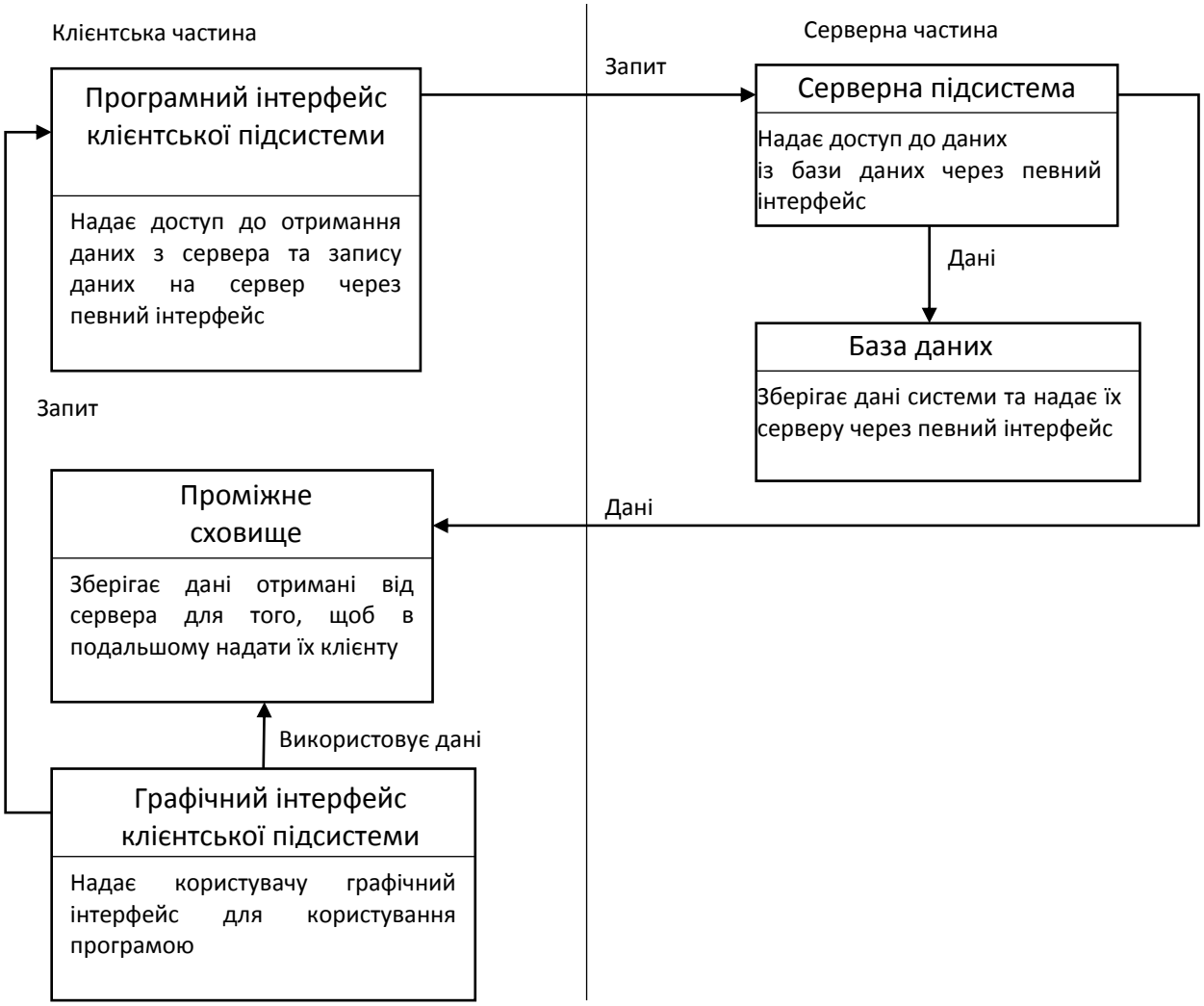


Рисунок 3.1 – Схема взаємодії компонентів системи

Як видно з даної діаграми, компоненти бувають 5 видів:

- Графічний інтерфейс клієнтської частини.
- Програмний інтерфейс клієнтської частини.
- Проміжне сховище.
- Сервер.

– База даних.

Графічний інтерфейс клієнтської частини – це компоненти, з якими безпосередньо взаємодіє користувач. В кінцевому вигляді вони представляють собою HTML-сторінки. В проміжному вигляді це зроблені за допомогою фреймворку React компоненти з певною логікою для покращення рівня інтерактивності. Компонент графічного інтерфейсу часто ще називають представленням.

Програмний інтерфейс клієнтської частини використовується графічним інтерфейсом. Він отримує запити на отримання певних даних, певним чином обробляє ці запити та робить нові адресовані серверу. Компонент програмного інтерфейсу часто називають контролером.

Серверна частина, в свою чергу, обробляє запити від програмного інтерфейсу та робить нові запити до бази даних. Серверні компоненти часто називають моделями.

База даних повертає ці дані в зворотному напрямку, спочатку серверу, який потім передає ці дані програмному інтерфейсу, а він записує ці дані у проміжне сховище.

Із проміжного сховища ці дані отримує компонент графічного інтерфейсу. Дані зберігаються у проміжному сховищі, а не напряму передаються від бази даних для того, щоб зменшити кількість запитів до бази даних. Проміжне сховище часто називають контейнером.

Основна бізнес-логіка відбувається на сервері. База даних і сервер існують в одиничному екземплярі. Але є багато серверних компонентів, які формують запити до бази даних. Також є багато проміжних сховищ, програмних та компонентів графічного інтерфейсу.

Проєкт використовує 3 фреймворки: Express для back-end, React для front-end та React Redux для зв’язку back-end та front-end. Програмні інтерфейси та проміжні сховища зроблені за допомогою React Redux. Графічний інтерфейс допомогою React. Серверні компоненти за допомогою Express.

Даний проєкт має наступні функціональні блоки:

– City – відповідає за інформацію про міста.
– Clinic – відповідає за інформацію про клініки.
– Country – відповідає за інформацію про країни.
– Department – відповідає за інформацію про відділення.
– Disease – відповідає за інформацію про діагнози.
– Employee – відповідає за інформацію про працівників.
– Job – відповідає за інформацію про посади.
– Medicament – відповідає за інформацію про медикаменти.
– Patient – відповідає за інформацію про пацієнтів.
– Presence – відповідає за інформацію про візити.
– Auth – відповідає за авторизацію користувачів.
– DiagnosStat – відповідає за статистику хвороб.
– SymptomStat – відповідає за статистику симптомів.
– TreatmentStat – відповідає за статистику схем лікування.
– ProcedureStat – відповідає за статистику процедур.
– MedicamentStat – відповідає за статистику медикаментів.
– Procedure – відповідає за інформацію про процедури.
– Room – відповідає за інформацію про кімнати.
– Symptom – відповідає за інформацію про симптоми.
– Treatment – відповідає за інформацію про схеми лікувань.
– User – відповідає за інформацію про користувачів.
– Diagnostic – відповідає за систему допомоги в діагностуванні хвороб.

В проєкті є наступні представлення:

– Register – відповідає за сторінку з реєстрацією.
– Login – відповідає за сторінку входу у систему.
– Clinics – відповідає за відображення клінік.
– Doctors – відповідає за відображення лікарів.
– Medicaments – відповідає за відображення медикаментів.
– Symptoms – відповідає за відображення симптомів.

– Procedures – відповідає за відображення процедур.

– Treatments – відповідає за відображення схем лікувань.

– Diagnosis – відповідає за відображення діагнозів.

– Visits – відповідає за відображення візитів.

– Diagnostics – відповідає за відображення системи діагностики.

– Countries – відповідає за відображення країн.

– Cities – відповідає за відображення міст.

– PersonalData – відповідає за відображення особистих даних користувача.

– StatisticsVisits – відповідає за відображення статистики візитів.

– StatisticsDiseases – відповідає за відображення статистики діагнозів.

– StatisticsTreatment – відповідає за відображення статистики схем лікувань.

– StatisticsMedicaments – відповідає за відображення статистики по медикаментах.

– StatisticsProcedures – відповідає за відображення статистики по процедурах.

– StatisticsSymptoms – відповідає за відображення статистики по симптомах.

Схеми всіх функціональних блоків програми наведені в додатках.

3.2 Особливості реалізації складових системи

Ключовими функціональними блоками даної системи є «візити» та «діагностика».

Сторінку візитів можна побачити на рисунках 3.2 – 3.3. На ній користувач може переглядати дані візитів до нього. А також перейти на сторінку з карткою пацієнта. Робота даної сторінки забезпечується функціональним модулем «Visits», який відповідає за візити і складається з контролера, контейнера та моделі.

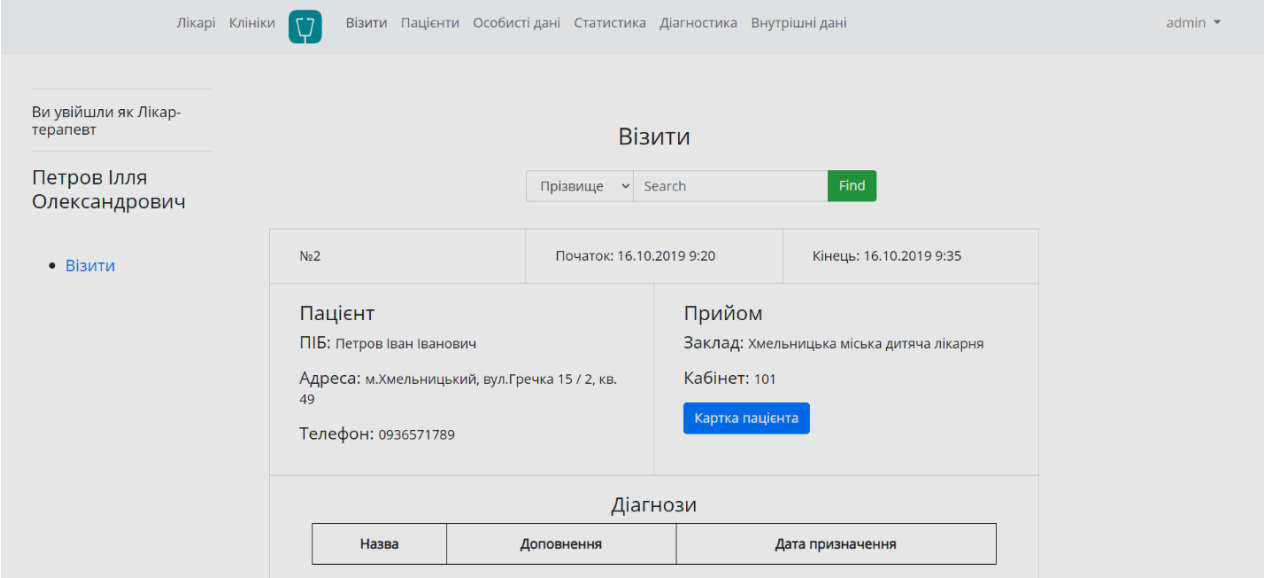


Рисунок 3.2 – Сторінка з візитами

Частина програмного коду контролера показана нижче.

```
export const getVisits = (user) => (dispatch) => {
  dispatch(setVisitsLoading());
  axios
    .get("/api/visits", {
      params: {
        ...user,
      },
    })
    .then((res) => {
      dispatch({
        type: GET_VISITS,
        payload: res.data,
      });
    })
    .catch((err) => {
      dispatch(
        returnErrors(err.response.data, err.response.status, "GET_VISITS ERROR")
      );
    });
};
```

Рисунок 3.3 – Сторінка з візитами

Сторінка з формою діагностики показана на рисунку 3.4.

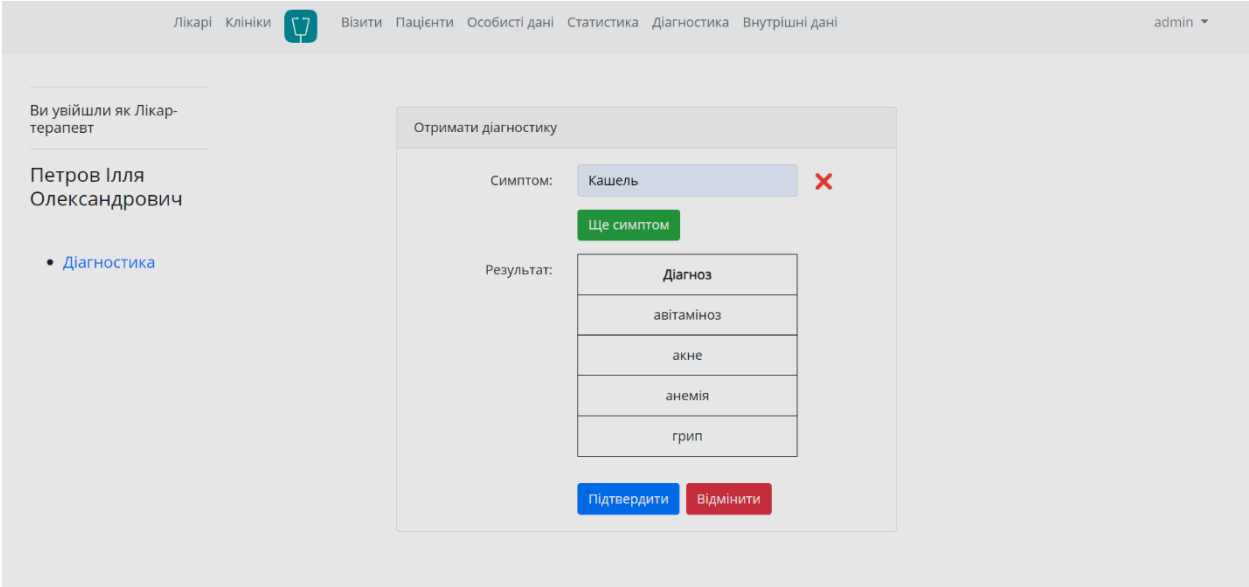


Рисунок 3.4 – Сторінка з формою діагностики

На ній користувач може ввести множину симптомів та отримати множину хвороб, які можуть характеризуватись даними симптомами. Робота даної сторінки забезпечується функціональним модулем «Diagnostics», який складається з контролера, контейнера та моделі.

Частина програмного коду контролера показана нижче.

```
export const generateDiagnostics = (symptoms) => (dispatch) => {
  dispatch(setDiagnosticsLoading());
  const config = {
    headers: {
      "Content-Type": "application/json",
    },
  };
  const body = JSON.stringify(symptoms);
  axios.post("/api/diagnostics/generate", body, config).then((res) => {
    dispatch({
      type: GENERATE_DIAGNOSTICS,
      payload: res.data,
    });
  });
};

export const getDiagnostics = () => (dispatch) => {
  dispatch(setDiagnosticsLoading());
  axios.get("/api/diagnostics").then((res) => {
    dispatch({
      type: GET_DIAGNOSTICS,
      payload: res.data,
    });
  });
};
```

Отже, схожим чином були створені всі компоненти системи.

3.3 Тестування інформаційної системи

Проведемо тестування роботи форми діагностики. Перевіримо чи повертає вона необхідні дані та як вона працює при некоректних даних.

Сторінку з формою діагностики показано на рисунку 3.5.

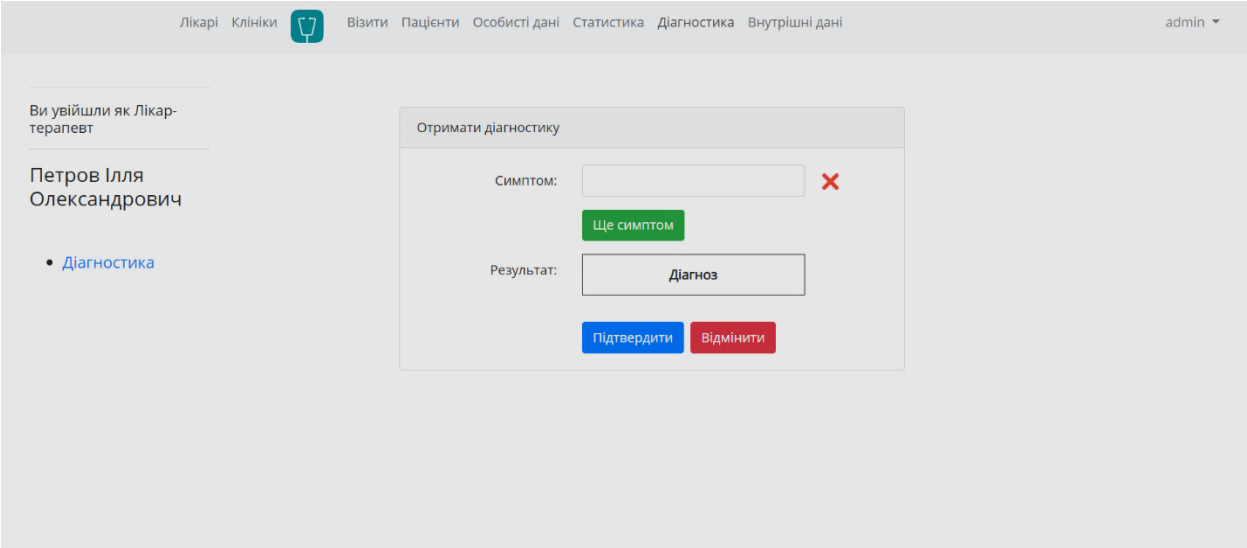


Рисунок 3.5 – Сторінка з формою діагностики

```
import diagnosticReducer, {
  initialState,
} from "../../../reducers/diagnosticReducer";
import * as types from "../../../actions/diagnostic/diagnosticTypes";

describe("diagnostics reducer", () => {
  it("should handle GENERATE_DIAGNOSTICS", () => {
    const action = {
      type: types.GENERATE_DIAGNOSTICS,
      payload: {
        diagnosis: [
          { id: 1, name: "авітаміноз" },
          { id: 2, name: "анемія" },
          { id: 6, name: "грип" },
        ],
      },
    };

    expect(diagnosticReducer(initialState, action)).toEqual({
      diagnostic: action.payload,
      innerData: [],
      loading: false,
    });
  });
});

```

Рисунок 3.6 – Код тест-кейса для перевірки роботи контейнера

На цій сторінці користувач може ввести набір симптомів, інформацію про які він, наприклад, отримав під час візиту до нього пацієнта. Отримує ж

користувач набір діагнозів, які можуть характеризуватись даними симптомами. Надалі користувач може використовувати отриману інформацію для прийняття рішення щодо лікування пацієнта.

Перевіримо чи правильно працює контейнер. Для цього напишемо тест-кейс, код якого зображений на рисунку 3.6.

Далі за допомогою бібліотеки тестів, яка доступна в React, ми запустимо даний тест-кейс та отримаємо результат, отриманий на рисунку 3.7.

З рисунку 3.7 видно, що тест-кейс був виконаний успішно. Це означає, що форма діагностики повертає правильні значення. Дана сторінка зображена на рисунку 3.8.

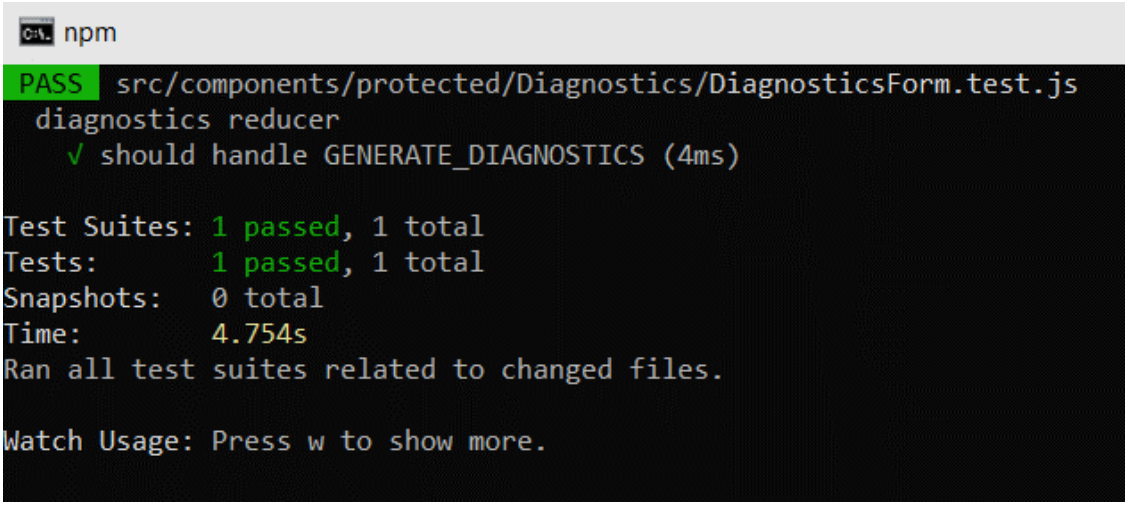


Рисунок 3.7 – Результат виконання тест-кейса

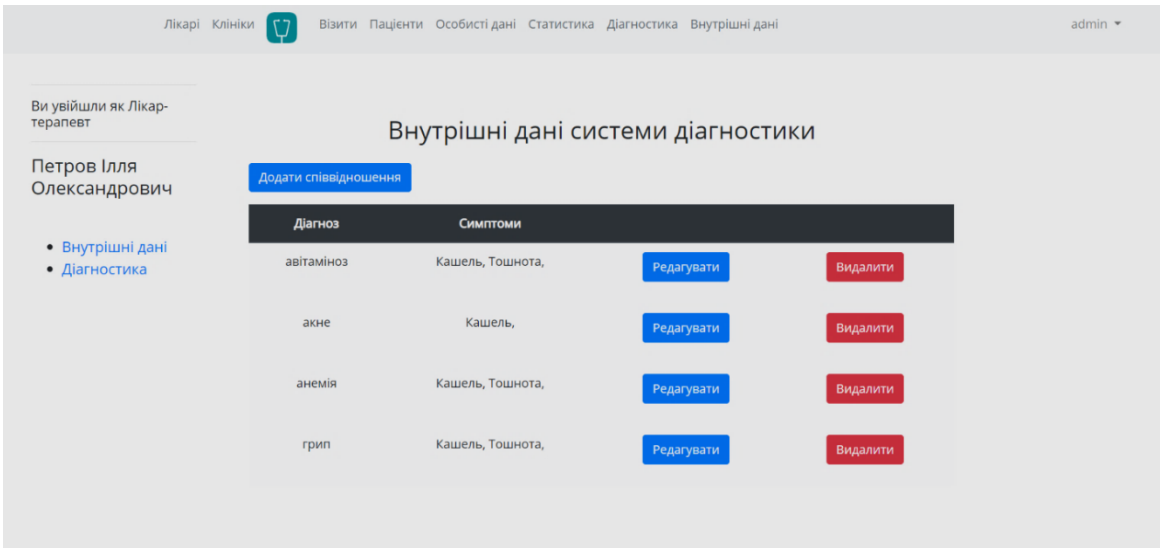


Рисунок 3.8 – Внутрішні дані системи діагностики

Далі ми перевіримо роботу сторінки для перегляду співвідношень діагнозів і симптомів. На цій сторінці адміністратор сайту може переглядати, редагувати, додавати та видаляти співвідношення, на основі яких працює система діагностики. Перевіримо чи правильно працює контейнер. Для цього напишемо тест-кейс, код якого зображений на рисунку 3.9. Далі за допомогою бібліотеки тестів, яка доступна в React, ми запустимо даний тест-кейс та отримаємо результат, отриманий на рисунку 3.10. З рисунку 3.10 видно, що тест-кейс був виконаний успішно. Це означає, що сторінки з внутрішніми даними системи діагностики повертає правильні значення.

```
27      it("should handle GET_DIAGNOSTICS", () => {
28        const action = {
29          type: types.GET_DIAGNOSTICS,
30          payload: {
31            diagnosis: [
32              {
33                diagnos: { id: 1, name: "авітаміноз" },
34                symptoms: [
35                  { id: 1, name: "Кашель" },
36                  { id: 2, name: "Тошнота" },
37                ],
38              },
39            ],
40          },
41        };
42
43        expect(diagnosticReducer(initialState, action)).toEqual({
44          diagnostic: {
45            diagnosis: [],
46          },
47          innerData: action.payload,
48          loading: false,
49        });
50      });
51    });
52
```

Рисунок 3.9 – Код тест-кейса для перевірки роботи контейнера

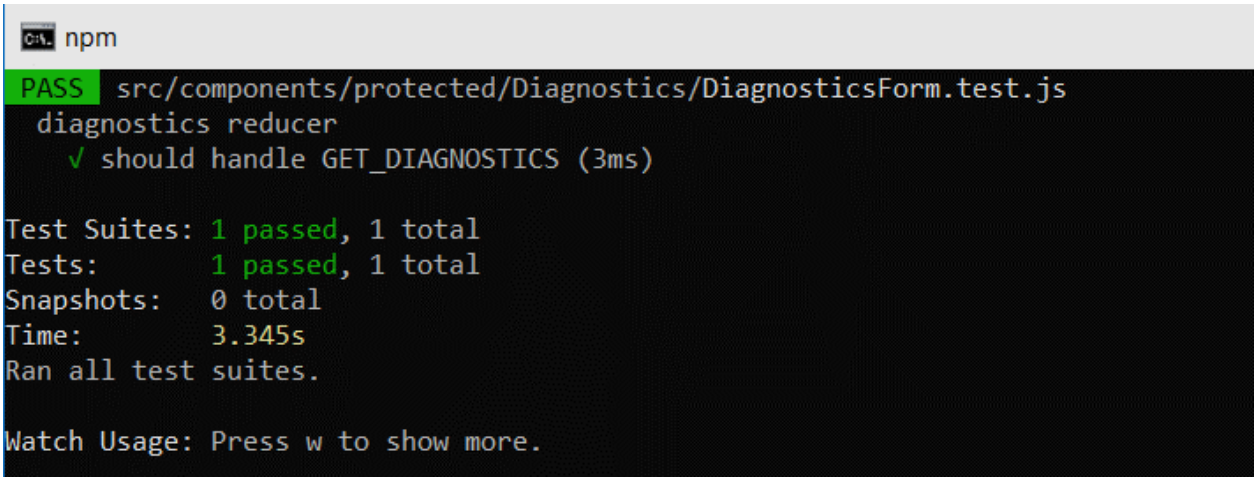


Рисунок 3.10 – Результат виконання тест-кейса

Отже, результати тестування показують, що всі функції системи працюють правильно.

3.4 Інструкція користувача

Перше, що бачить користувач, зайшовши на сайт – це стартова сторінка, зображена на рисунку 3.11.

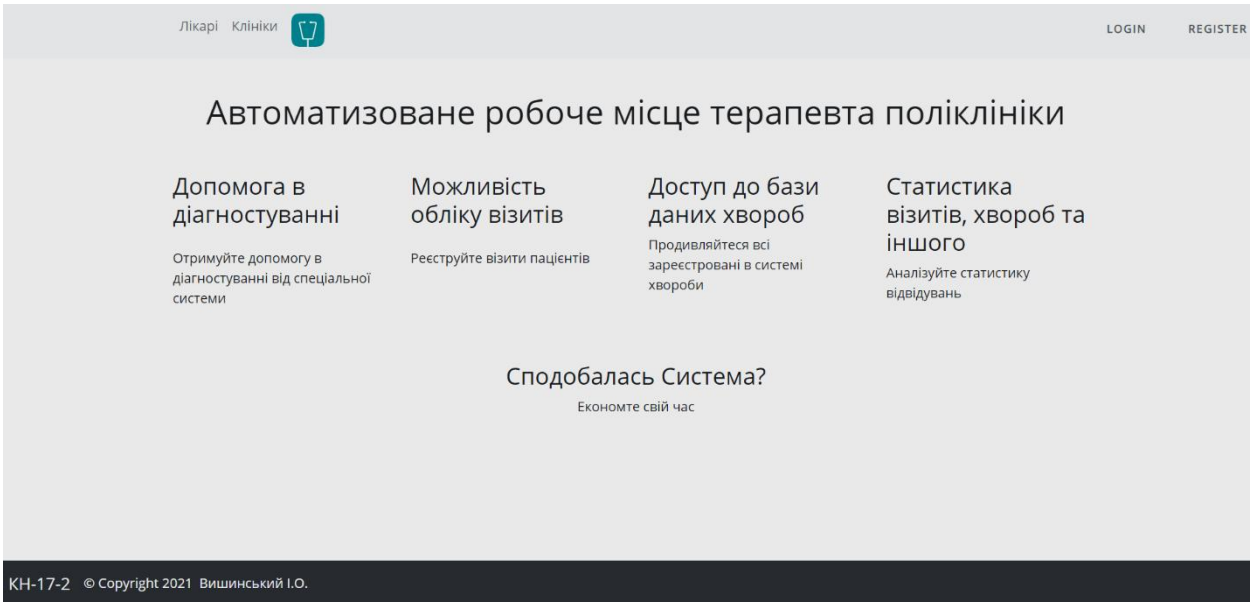


Рисунок 3.11 – Стартова сторінка

Далі користувач може зайти сторінку з інформацією про лікарів, де побачить інформацію про лікарів, які зареєстровані в системі. Ця сторінка зображена на рисунку 3.12.

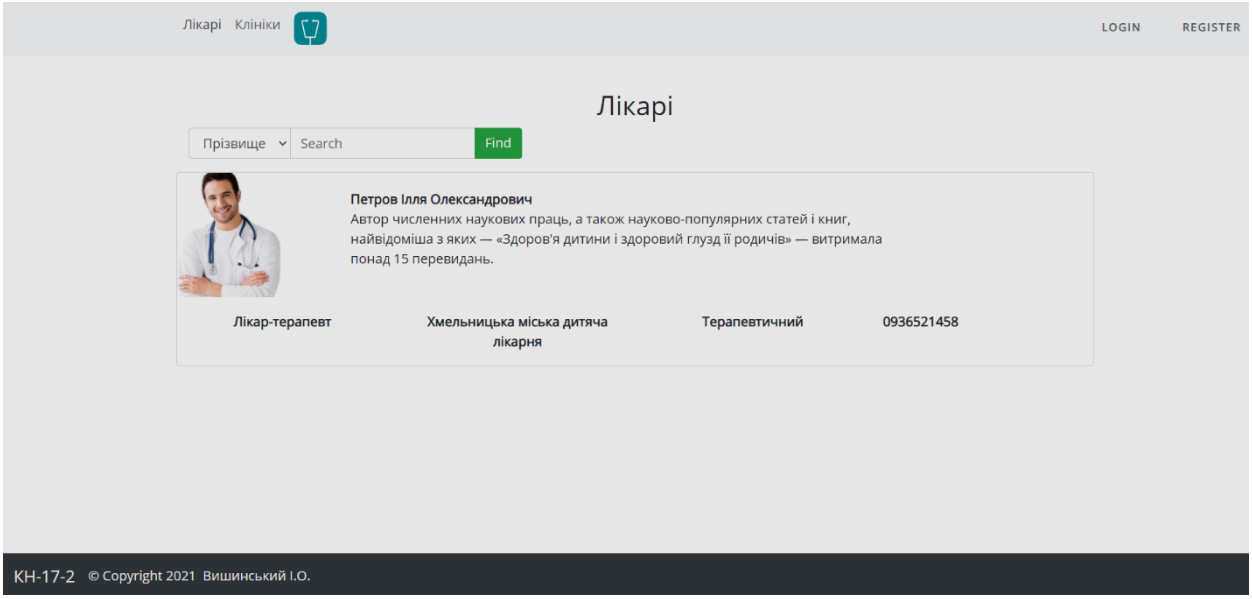


Рисунок 3.12 – Сторінка з інформацією про лікарів

Також зі стартової сторінки користувач може перейти на сторінку з інформацією про клініки, доступні в системі. Дана сторінка зображена на рисунку 3.13.

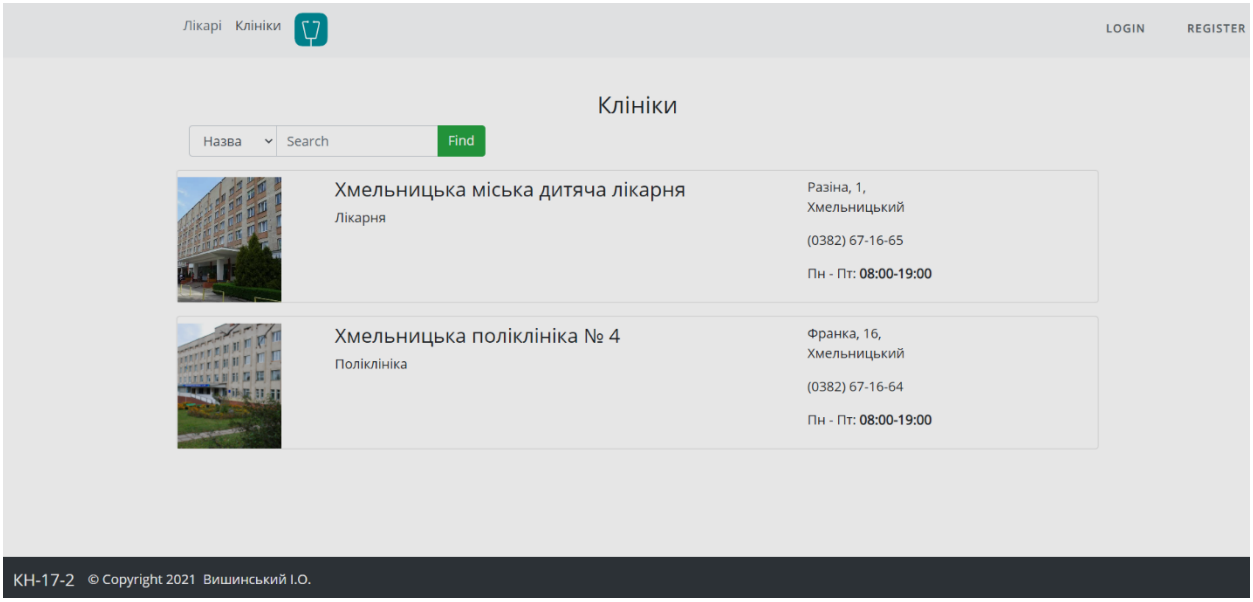


Рисунок 3.13 – Сторінка з інформацією про клініки

Крім цього, зі стартової сторінки користувач може перейти на сторінку з логіном або реєстрацією. Сторінка з реєстрацією зображена на рисунку 3.14.

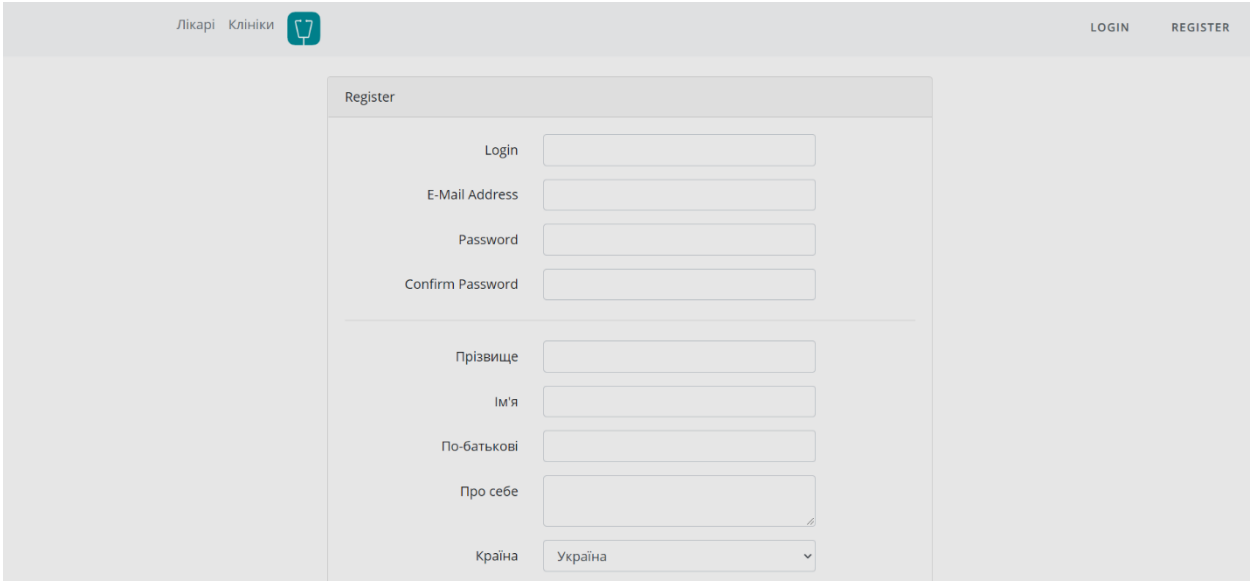


Рисунок 3.14 – Сторінка з реєстрацією для користувача

Сторінка з логіном для користувача зображена на рисунку 3.15.

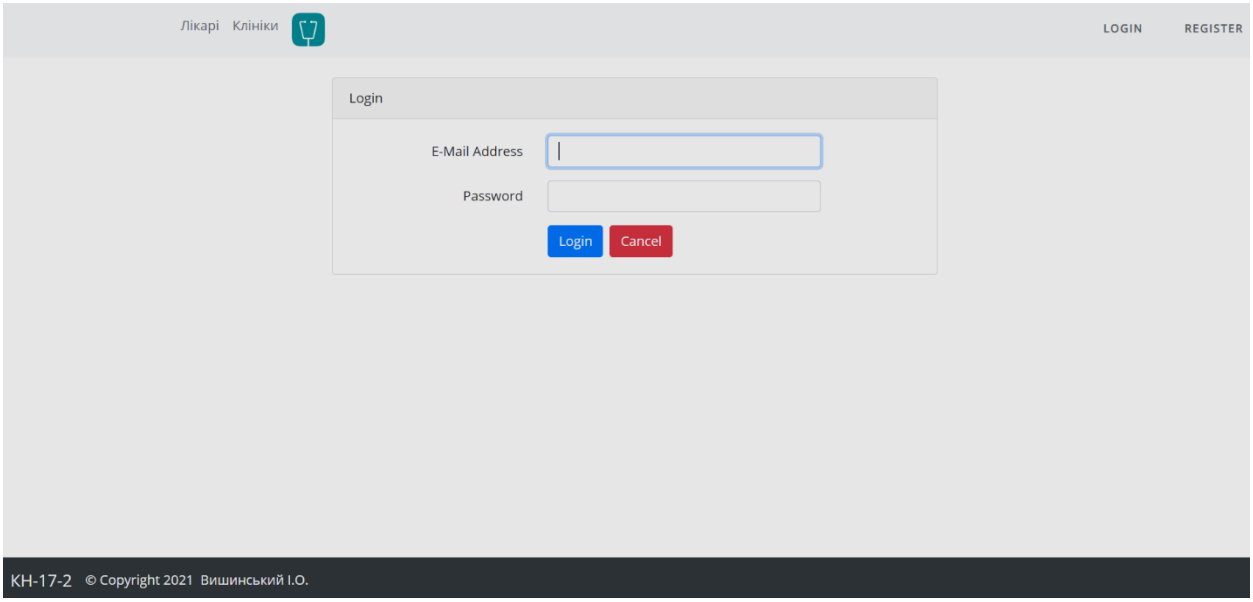


Рисунок 3.15 – Сторінка з логіном для користувача

Здійснивши авторизацію, користувач може здійснювати навігацію по сторінкам, які доступні лише авторизованим користувачам. Сторінка з візитами зображена на рисунках 3.16 – 3.17.

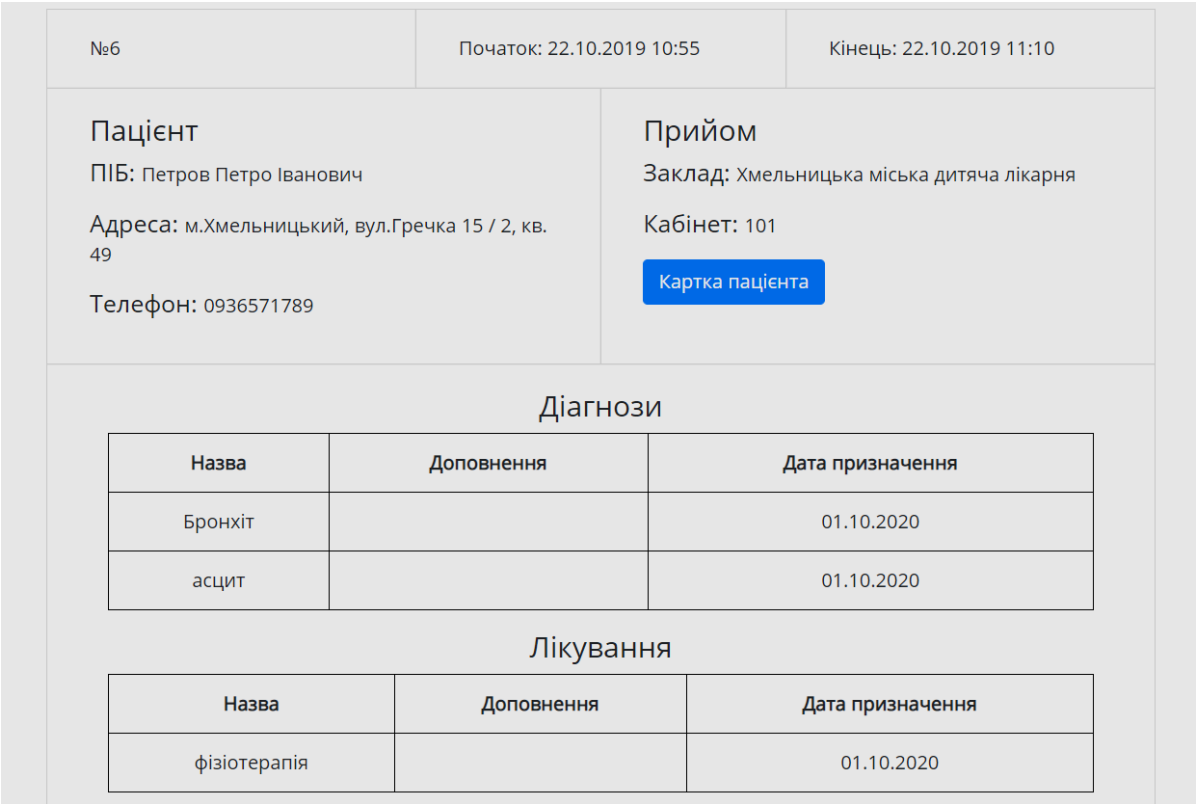


Рисунок 3.16 – Сторінка з візитами

Рисунок 3.17 – Сторінка з візитами

На сторінці з візитами користувач може перейти на сторінку з карткою пацієнта, натиснувши кнопку «Картка пацієнта». А також він може здійснювати пошук візитів, натиснувши кнопку «Find». Сторінка з карткою пацієнта зображена на рисунках 3.18 – 3.19.

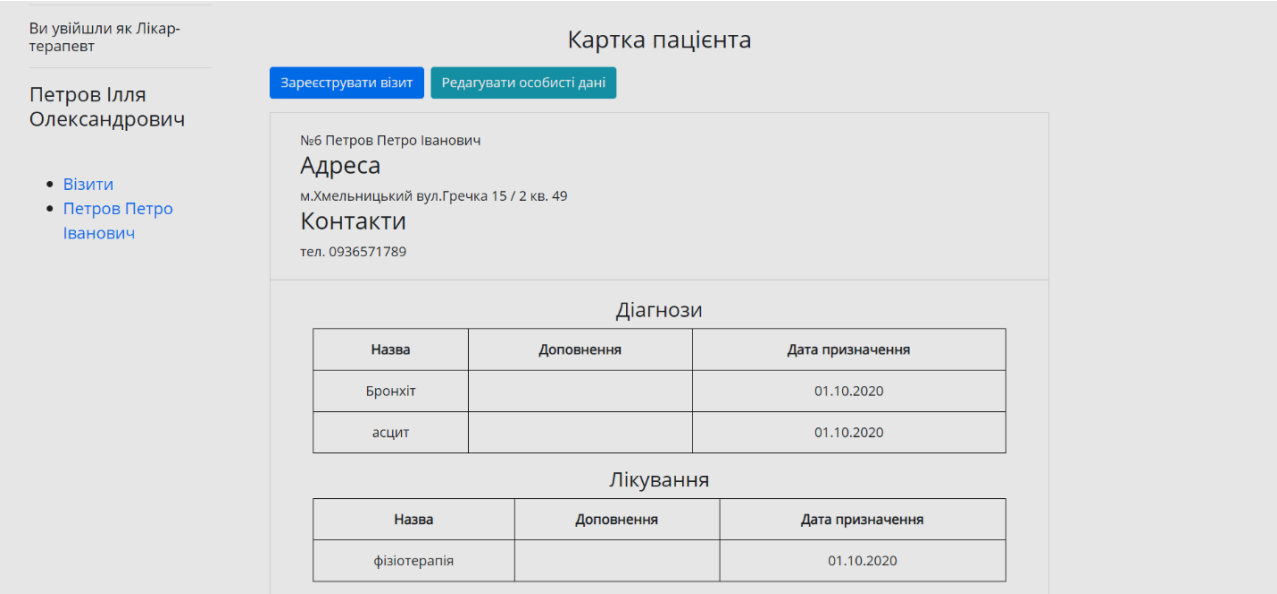


Рисунок 3.18 – Сторінка з карткою пацієнта

Рисунок 3.19 – Сторінка з карткою пацієнта

На даній сторінці користувач може продивлятися інформацію про раніше призначені пацієнту діагнози, медикаменти, процедури, схеми лікувань, його особисту інформацію. А також може реєструвати нові візити даного пацієнта та редагувати його особисті дані.

Також з домашньої сторінки користувач може перейти на сторінку з його особистою інформацією, зображену на рисунку 3.20. На даній сторінці користувач може продивлятися та редагувати свою особисту інформацію. Ще з домашньої сторінки користувач може перейти на сторінку зі статистикою своєї роботи. Дана сторінка зображена на рисунку 3.21.

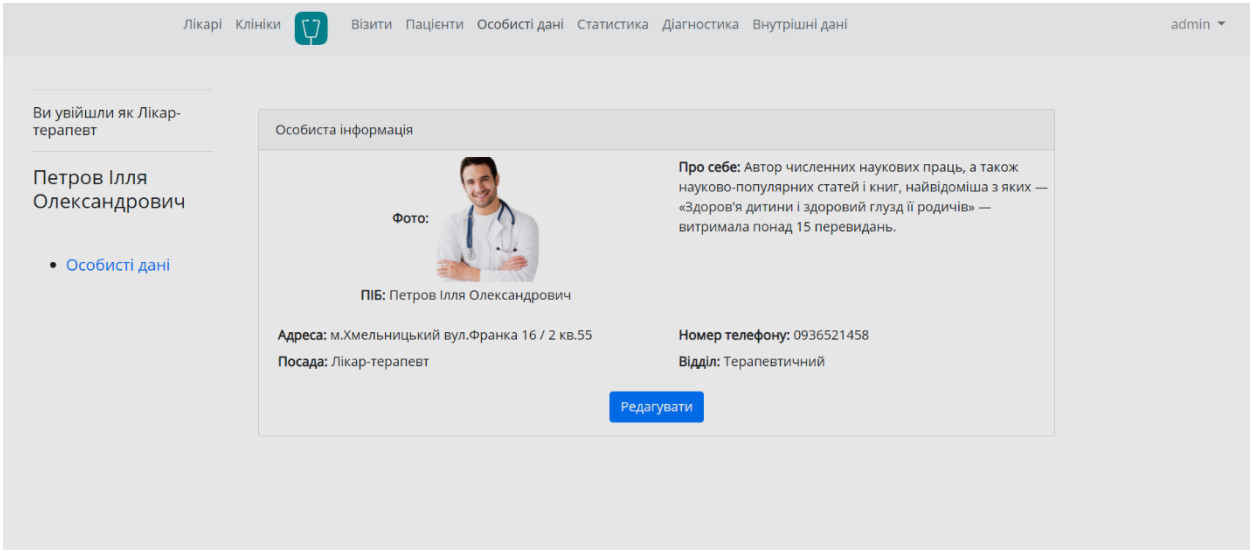


Рисунок 3.20 – Сторінка з особистими даними користувача

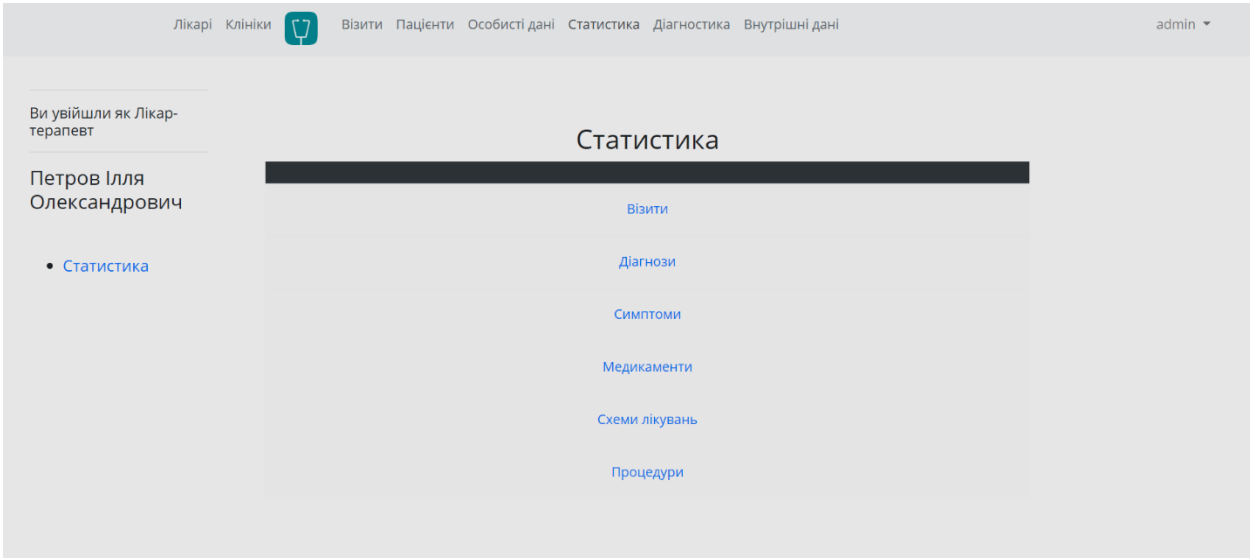


Рисунок 3.21 – Сторінка зі статистикою робити користувача

З цієї сторінки користувач, натискаючи відповідні кнопки, може побачити статистику по відповідним даним. Фрагмент сторінки зі статистикою візитів зображена на рисунку 3.22.

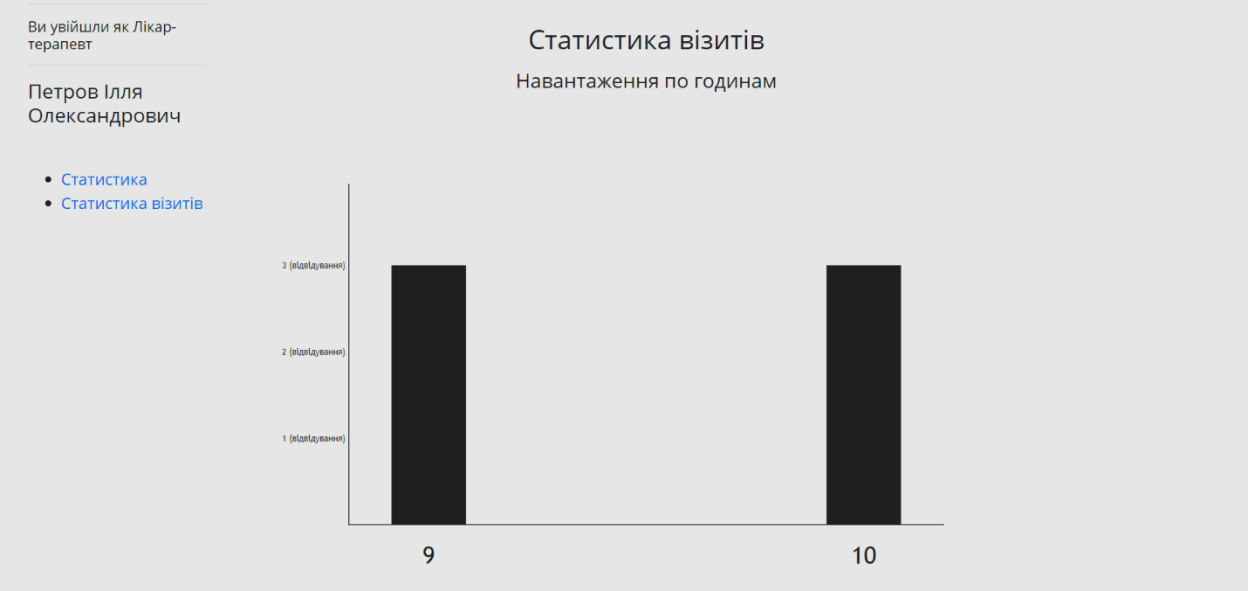


Рисунок 3.22 – Сторінка зі статистикою візитів до користувача

Ще з домашньої сторінки користувач з правами адміністратора може перейти на сторінку з внутрішніми даними системи, де побачить посилання на відповідні сторінки, де він зможе переглядати, додавати та іноді редагувати дані із системи. Сторінка з посиланнями зображена на рисунку 3.23.

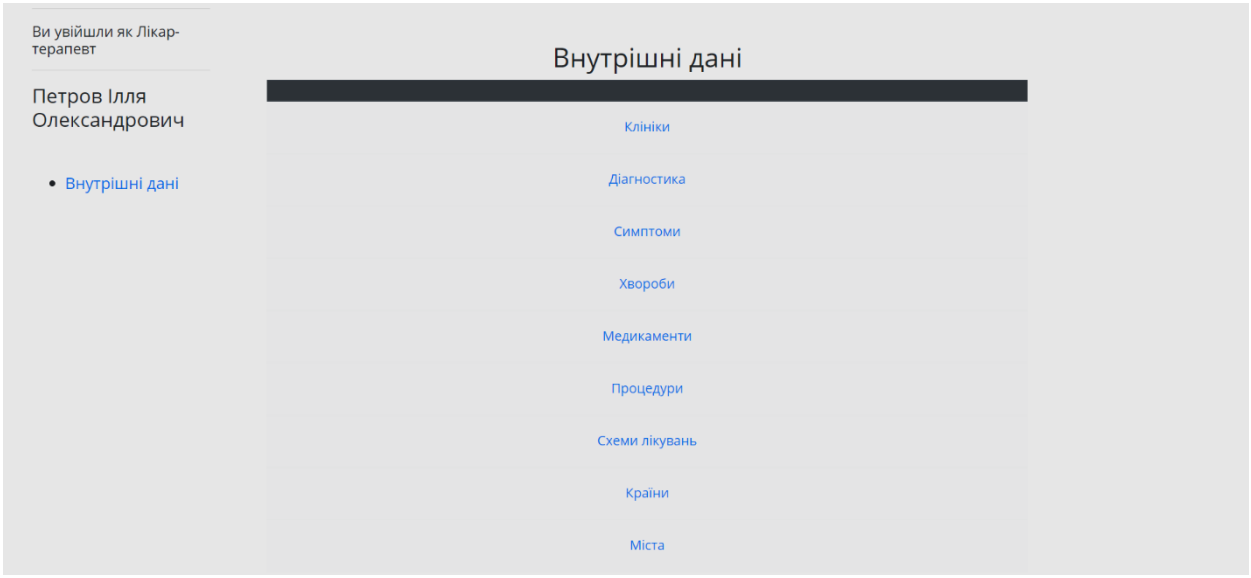


Рисунок 3.23 – Сторінка зі внутрішніми даними системи

З цієї сторінки користувач може перейти на сторінку з внутрішніми даними системи. Сторінка з даними про клініки зображена на рисунку 3.24.

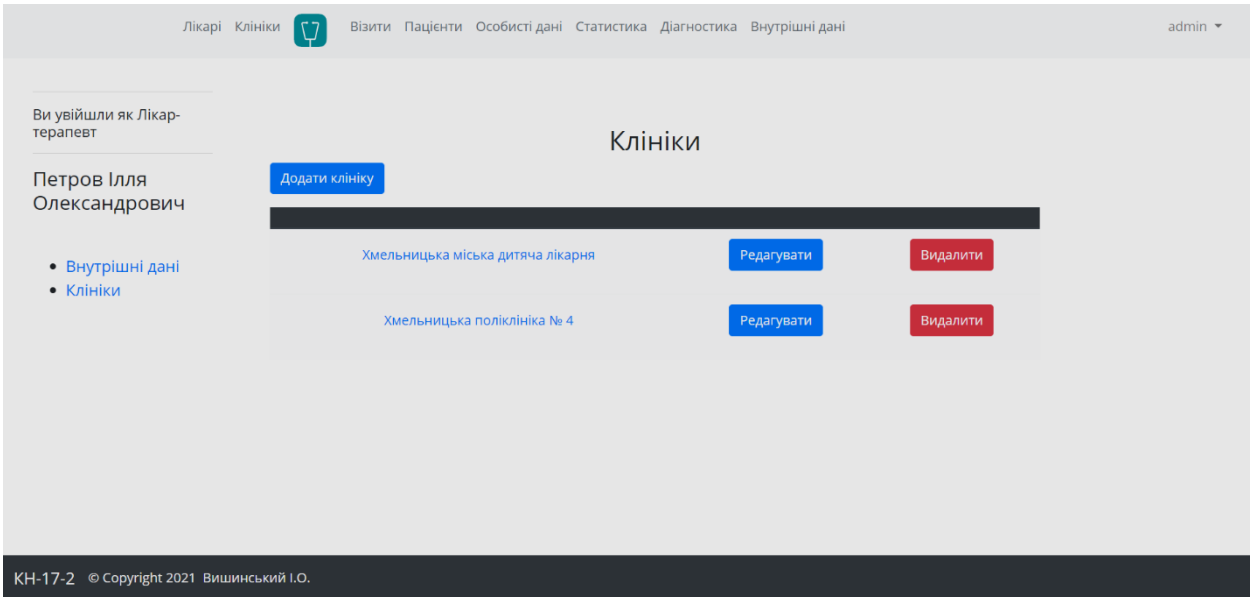


Рисунок 3.24 – Сторінка з внутрішніми даними про клініки

Також з домашньої сторінки користувач може перейти на сторінку з картками пацієнтів. що зображена на рисунку 3.25.

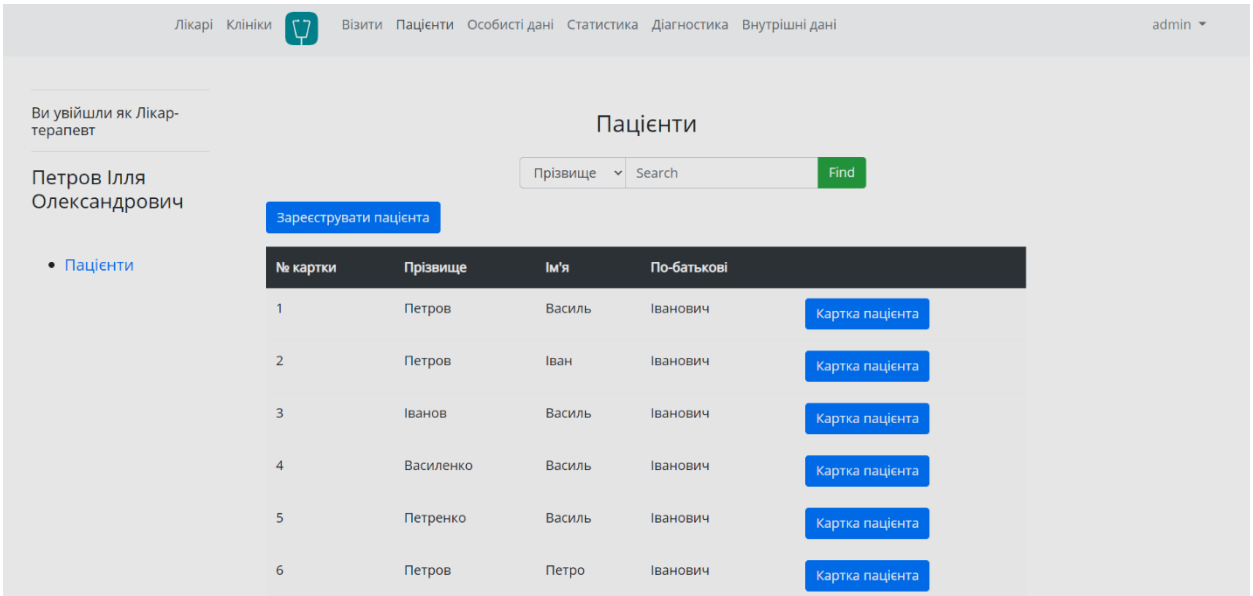


Рисунок 3.25 – Сторінка з картками пацієнтів

Звідси користувач може перейти на сторінку з карткою певного пацієнта, а також зареєструвати нового пацієнта. Сторінка з карткою пацієнта зображена на рисунках 3.26 – 3.27.

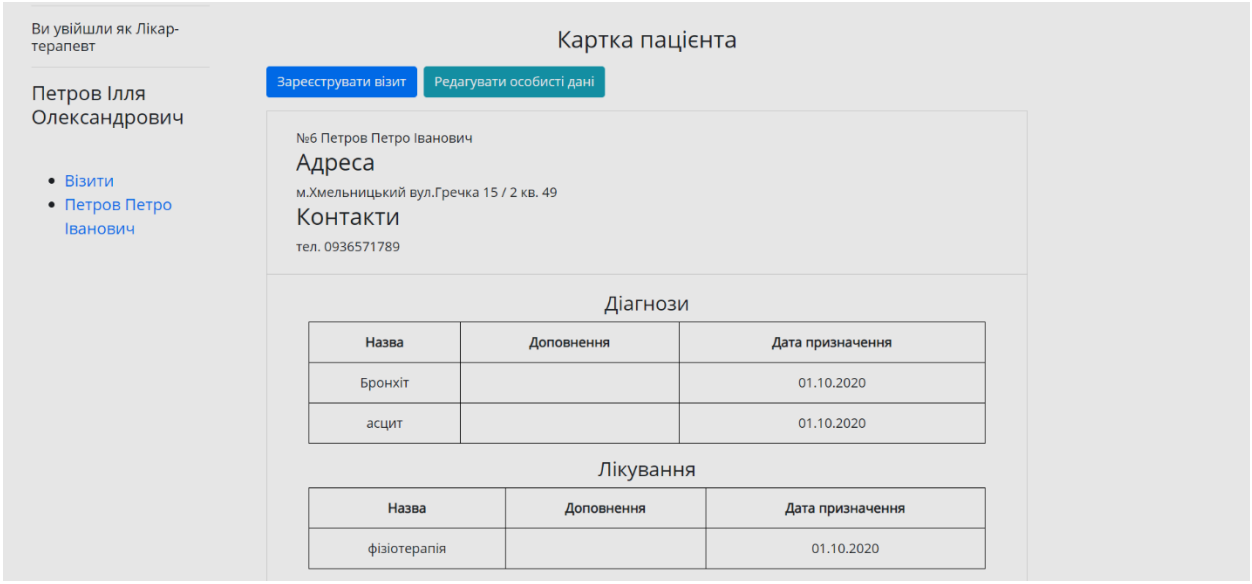


Рисунок 3.26 – Сторінка з карткою пацієнта

Рисунок 3.27 – Картка пацієнта

На даній сторінці користувач може продивлятися інформацію про раніше призначені пацієнту діагнози, медикаменти, процедури, схеми лікувань, його

особисту інформацію. А також може реєструвати нові візити даного пацієнта та редагувати його особисті дані.

Користувач може здійснювати редагування, додавання та видалення певної інформації. Форми для додавання та редагування зроблені по одному шаблону, а, отже, виглядають майже однаково. Приклад форми додавання інформації зображено на рисунках 3.28-3.30.

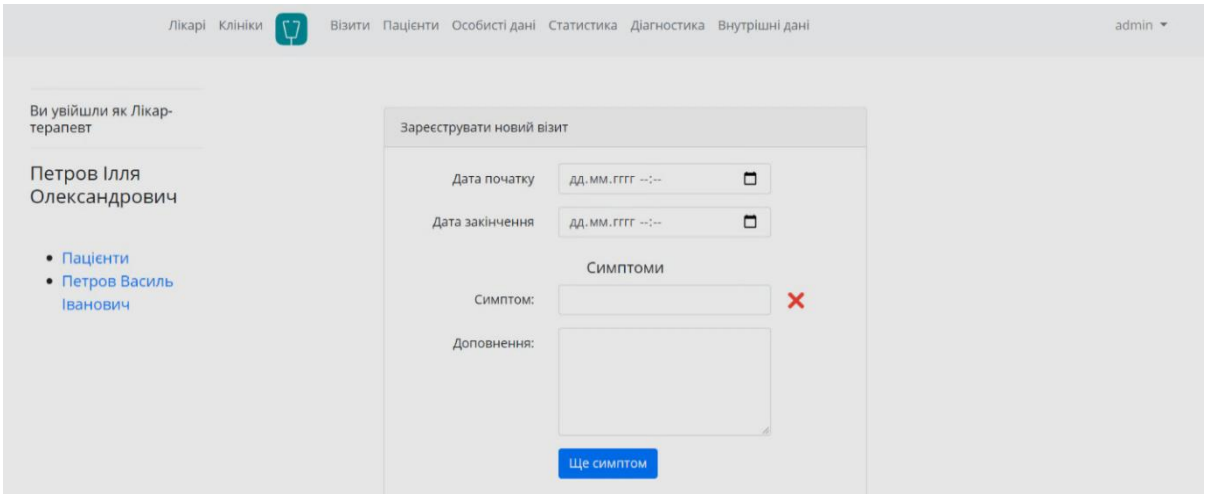


Рисунок 3.28 – Форма додавання нового візиту

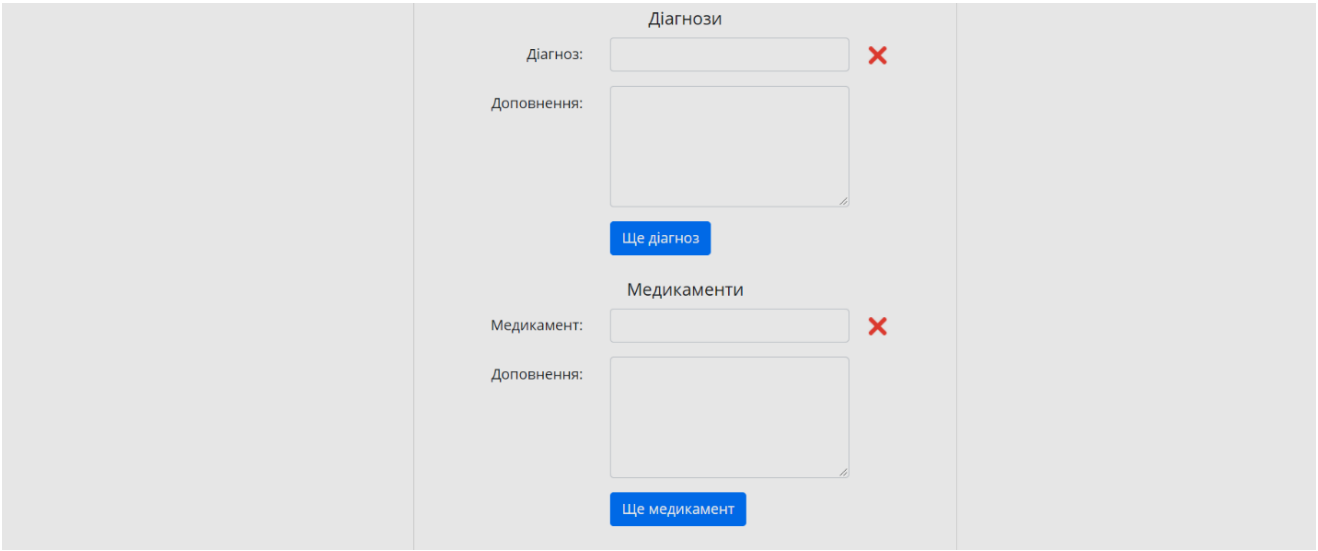


Рисунок 3.29 – Форма додавання нового візиту

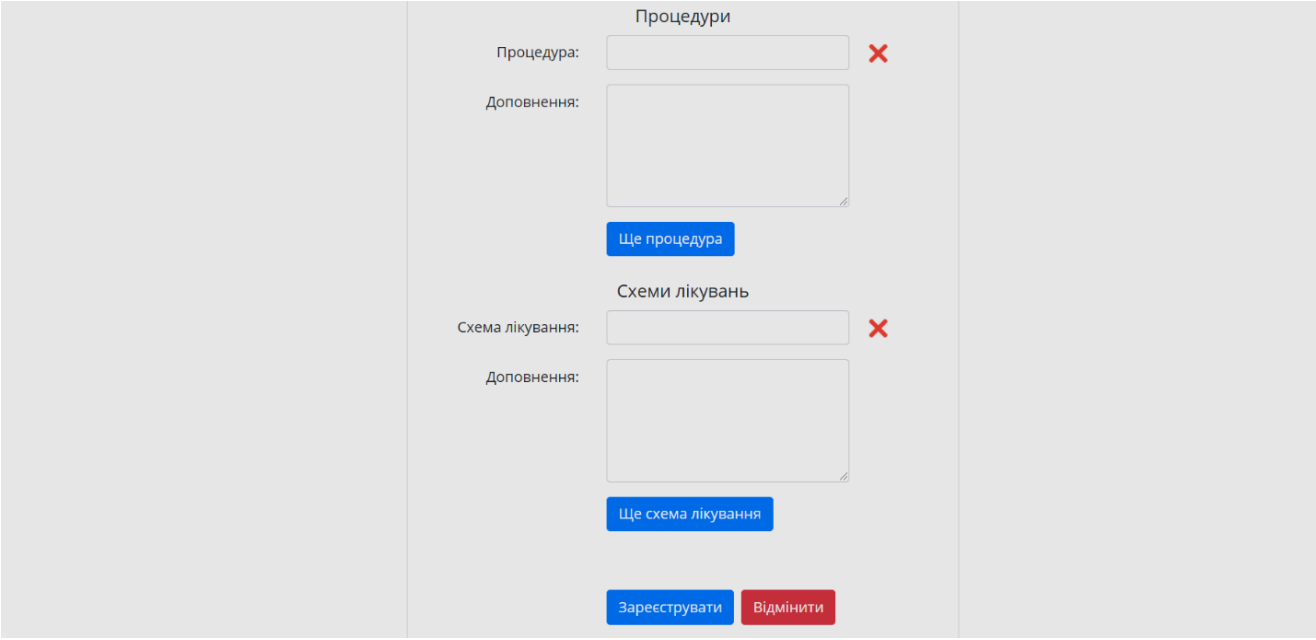


Рисунок 3.30 – Форма додавання нового візиту

Приклад редагування інформації зображено на рисунку 3.31.

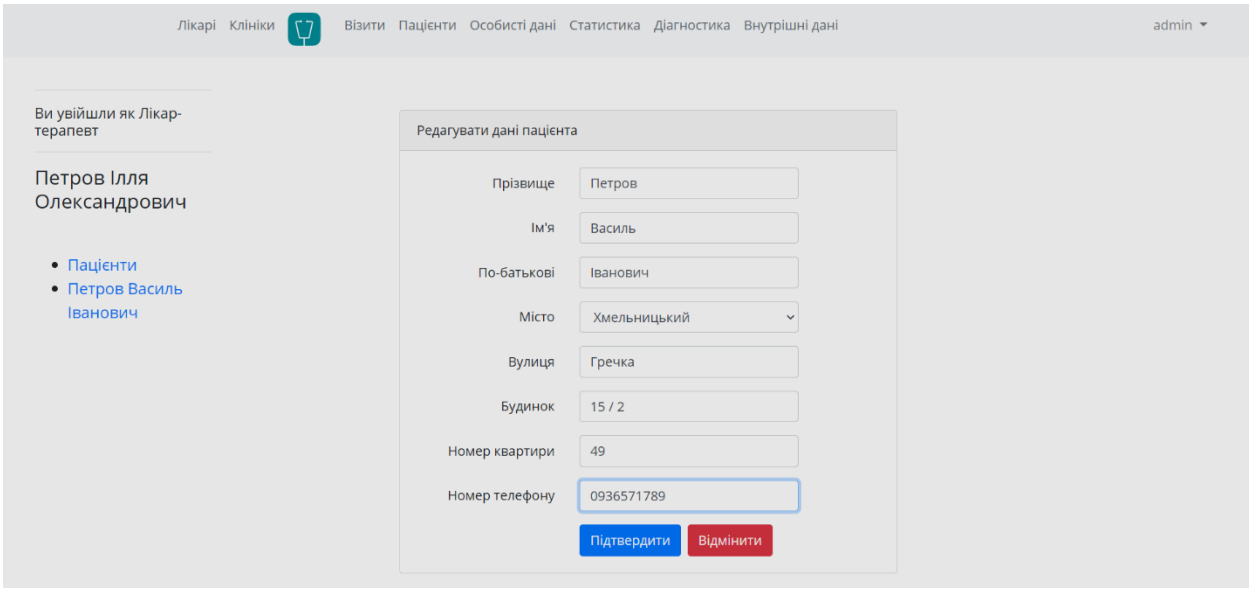


Рисунок 3.31 – Форма редагування особистих даних пацієнта

На сторінці «Діагностика» користувач може вписати набір певних симптомів та, натиснувши відповідну кнопку, отримати набір діагнозів, які характеризуються даними симптомами. Цю сторінку зображено на рисунку 3.32.

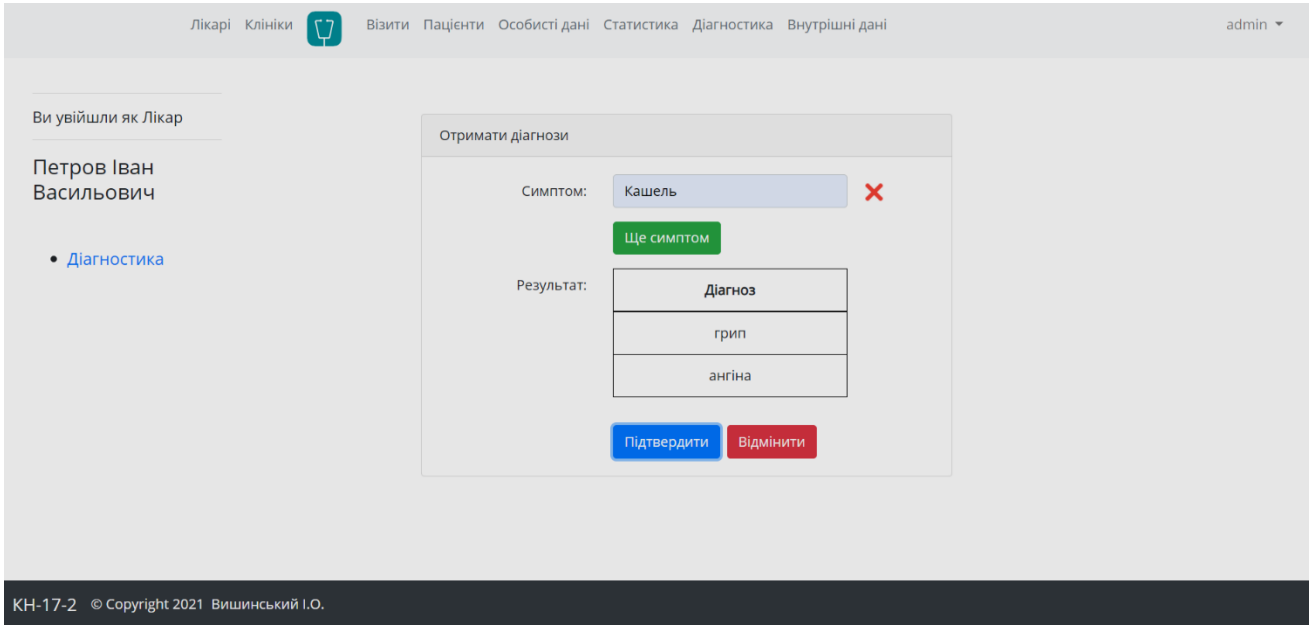


Рисунок 3.32 – Сторінка «Діагностика»

Отже, користування сайтом є інтуїтивно зрозумілим та не вимагає додаткових знань, умінь та навичок, окрім необхідних для звичайного користування комп’ютером.

3.5 Вимоги до розгортання інформаційної системи

Мінімальні вимоги до розгортання клієнтської частини.

Вимоги до апаратних засобів:

– Процесор: Intel Core I3.

– ОЗУ: 1 GB DDR3.

– Мережевий контролер: 4 * 1 Gb Ethernet.

– Жорсткий диск: HDD (128GB).

Вимоги до програмних засобів:

Операційна система: Linux (Ubuntu Server). Version: 16.0 +.

Сервер: Nginx. Version: 1.12 +.

Програмна платформа: Node.js. Version: 15.1 +.

Менеджер пакетів: npm. Version: 7.7.4 +.

Пакети:

– Testing-library/jest-dom: v4.2.4.
– Testing-library/react: v9.5.0.
– Testing-library/user-event: v7.2.1.
– Aws-sdk: v2.786.0.
– Axios: v0.21.1.
– Buffer: v6.0.1.
– Config: v3.3.2.
– Moment: v2.29.1.
– React: v16.13.1.
– React-dom: v16.13.1.
– React-redux: v7.2.1.
– React-router-dom: v5.2.0.
– React-scripts: v3.4.4.
– Redux: v4.0.5.
– Redux-thunk: v2.3.0.
– Victory: v35.5.0.

Мінімальні вимоги до розгортання клієнтської частини.

Вимоги до апаратних засобів:

– Процесор: Intel Core I3.
– ОЗУ: 4 GB DDR3.
– Мережевий контролер: 4 * 1 Gb Ethernet.
– Жорсткий диск: HDD (128GB).

Вимоги до програмних засобів:

Операційна система: Linux (Ubuntu Server). Version: 16.0 +.

Сервер: Node.js. Version: 13 +.

Програмна платформа: Node.js. Version: 15.1 +.

Менеджер пакетів: npm. Version: 7.7.4 +.

Пакети:

– Bcryptjs: v2.4.3.

– Concurrently: v5.3.0.

– Config: v3.3.2.

– Dotenv: v8.2.0.

– Express: v4.17.1.

– Jsonwebtoken: v8.5.1.

– Mysql2: v2.2.5.

Мінімальні вимоги до браузера:

– Підтримка HTML 5.

– Підтримка CSS 3.

– Підтримка ES 2015.

Мінімальні вимоги до користувача:

– Для роботи вимагається підключення до Інтернету.

– Сайт не створений для перегляду на мобільних пристроях або інших пристроях з маленьким розміром екрану.

Висновки

В результаті виконання даної КРБ було розроблено автоматизовану систему допомоги в діагностуванні хвороб терапевтом на платформі JavaScript, використовуючи фреймворки Express, React, Redux і СКБД MySQL. Всі вимоги, поставленні до роботи, були виконані, а саме була створена база даних та застосунок, який її використовує. Створена система виконує наступні функції:

– Робота з відкритою інформацією (виведення інформації про лікарів та поліклініки, зареєстровані в системі).

– Робота з візитами (додавання симптомів пацієнта, діагнозів, направлень на процедури, рецептів на медикаменти).

– Робота з картками пацієнтів (додавання симптомів пацієнта, діагнозів, направлень на процедури, рецептів на медикаменти, редагування його особистої інформації).

– Робота з особистими даними (реєстрація в системі, авторизація, редагування даних профілю).

– Робота з внутрішніми даними системи (хвороби, симптоми, медикаменти, процедури, працівники, пацієнти, палати).

– Робота з допомогою в діагностуванні (отримання діагнозів по симптомах, додавання нових даних в систему діагностики).

– Робота зі статистикою роботи терапевта (по симптомах пацієнта, діагнозах, направленнях на процедури, рецептах на медикаменти).

Дана система може бути використана для автоматизації роботи терапевта поліклініки. Її можна вдосконалити, додавши можливість генерування направлень, можливість автоматизовано отримувати дані аналізів від лікарень, а також автоматизовану відправку направлень лікарням.

Перелік посилань

1. sum.in.ua. Академічний тлумачний словник української мови. URL: http://sum.in.ua/s/poliklinika

2. Шемшученко Ю. С. Юридична енциклопедія. 1998. № ISBN 966-749-200-1. С. 120.

3. zakon.rada.gov.ua. Термінологія законодавства (станом на 26.03.2021). URL: https://zakon.rada.gov.ua/laws/term/19997

4. pharmencyclopedia. Фармацевтична енциклопедія. URL: https://www.pharmencyclopedia.com.ua/article/2525/diagnoz

5. clinpharm. Теоретичне обґрунтування та реалізація сучасних принципів оцінки еквівалентності лікарських засобів в Україні (метод. рекомендації). URL: https://clinpharm.nuph.edu.ua/upload/files/TOA/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BA%D0%B0-%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BD%D1%82-%D1%81%D0%B0%D0%B9%D1%82-%D0%9D%D0%A4%D0%B0%D0%A3.pdf

6. dec.gov.ua. Перелік медичних процедур (послуг) та хірургічних операцій що виконуються лікарями КЗПО. Наказ МОЗ України Від 14.02.2007 р. №67. URL: https://www.dec.gov.ua/wp-ontent/uploads/2019/12/2007_67_klasifikator_mp_xo.pdf

7. pharmencyclopedia. Фармацевтична енциклопедія. URL: https://www.pharmencyclopedia.com.ua/article/2156/terapiya

8. esu. Енциклопедія сучасної України. URL: http://esu.com.ua/search_articles.php?id=55462

9. Шмиг Р. А., Боярчук В. М., Добрянський І. М., Барабаш В. М. Термінологічний словник-довідник з будівництва та архітектури. 2010. № ISBN 978-966-7407-83-4. С. 115.

10. pharmencyclopedia. Фармацевтична енциклопедія. URL: https://www.pharmencyclopedia.com.ua/article/2156/terapiya

11. Ковальова О. М., Сафаргаліна-Корнілова Н. А. Пропедевтика внутрішньої медицини. 2010. № ISBN 978-617-505-094-1. 720 с.

12. Щуліпенко І. М. Пропедевтика внутрішньої медицини: загальна діагностика і семіотика. 2008. № ISBN 978-966-10-0011-6. С.123-138, с.308.

13. Medics. Головна сторінка. URL: https://medics.ua/

14. dpatient.bravosoft.org. Головна сторінка. URL: http://dpatient.bravosoft.org/

15. Medcard24. Головна сторінка. URL: https://patient.medcard24.net/

16. Олищук А. В. Розробка Web-додатків на PHP 5. 2006. С. 352.

17. Silberschatz Abraham. Operating System Concepts. 1994. С. 97.

18. Calltouch. Що таке мобільний додаток і навіщо він може знадобитись. URL: https://www.calltouch.ru/glossary/chto-takoe-mobilnoe-prilozhenie-i-zachem-ono-mozhet-potrebovatsya/

19. World Wide Web Consortium. Головна сторінка. URL: https://www.w3.org/

20. Microsoft. Dotnet. URL: https://dotnet.microsoft.com/

21. Java. Головна сторінка. URL: https://www.java.com/ru/

22. PHP. Manual. URL: https://www.php.net/manual/ru/intro-whatis.php

23. Developer.Mozilla. JavaScript. URL: https://developer.mozilla.org/ru/docs/Web/JavaScript

24. CppReference. Головна сторінка. URL: https://en.cppreference.com/w/

25. Docs.Microsoft. Csharp. URL: https://docs.microsoft.com/en-us/dotnet/csharp/

26. Docs.Microsoft. Visual Basic. URL: https://docs.microsoft.com/ru-ru/visualstudio/get-started/visual-basic/tutorial-console?view=vs-2019

27. Microsoft. SQL-SERVER 2019. URL: https://www.microsoft.com/ru-ru/sql-server/sql-server-2019

28. Docs.Microsoft. ASP.NET CORE. URL: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0

29. Microsoft. Windows. URL: https://www.microsoft.com/uk-ua/windows

30. Microsoft. Головна сторінка. URL: https://www.microsoft.com/uk-ua

31. VisualStudio.Microsoft. Головна сторінка. URL: https://visualstudio.microsoft.com/ru/

32. Tproger. Frontend-backend-interaction. URL: https://tproger.ru/translations/frontend-backend-interaction/

33. FructCode. Features-of-popular-frameworks-html-css-php-and-python-frameworks. URL: https://fructcode.com/ru/blog/features-of-popular-frameworks-html-css-php-and-python-frameworks/

34. Reactjs. Головна сторінка. URL: https://ru.reactjs.org/

35. Angular. Головна сторінка. URL: https://angular.io/

36. Developer.Mozilla. HTML. URL: https://developer.mozilla.org/ru/docs/Web/HTML

37. Developer.Mozilla. JSON. URL: https://developer.mozilla.org/ru/docs/Learn/JavaScript/Objects/JSON

38. Linux. Головна сторінка. URL: https://www.linux.org.ru/

39. PostgreSQL. Головна сторінка. URL: https://www.postgresql.org/

40. Habr. MVC для веб. URL: https://habr.com/ru/post/181772/

41. Developer.Mozilla. DOM. URL: https://developer.mozilla.org/ru/docs/Web/API/Document_Object_Model/Introduction

42. Express. Головна сторінка. URL: https://expressjs.com/ru/

43. Redux. Головна сторінка. URL: https://redux.js.org/

44. Habr. Архітектура REST. URL: https://habr.com/ru/post/38730/

45. Dev.by. Що таке API простою мовою. URL: https://dev.by/news/chto-takoe-api-prostym-yazykom

46. Nodejs. Головна сторінка. URL: https://nodejs.org/uk/

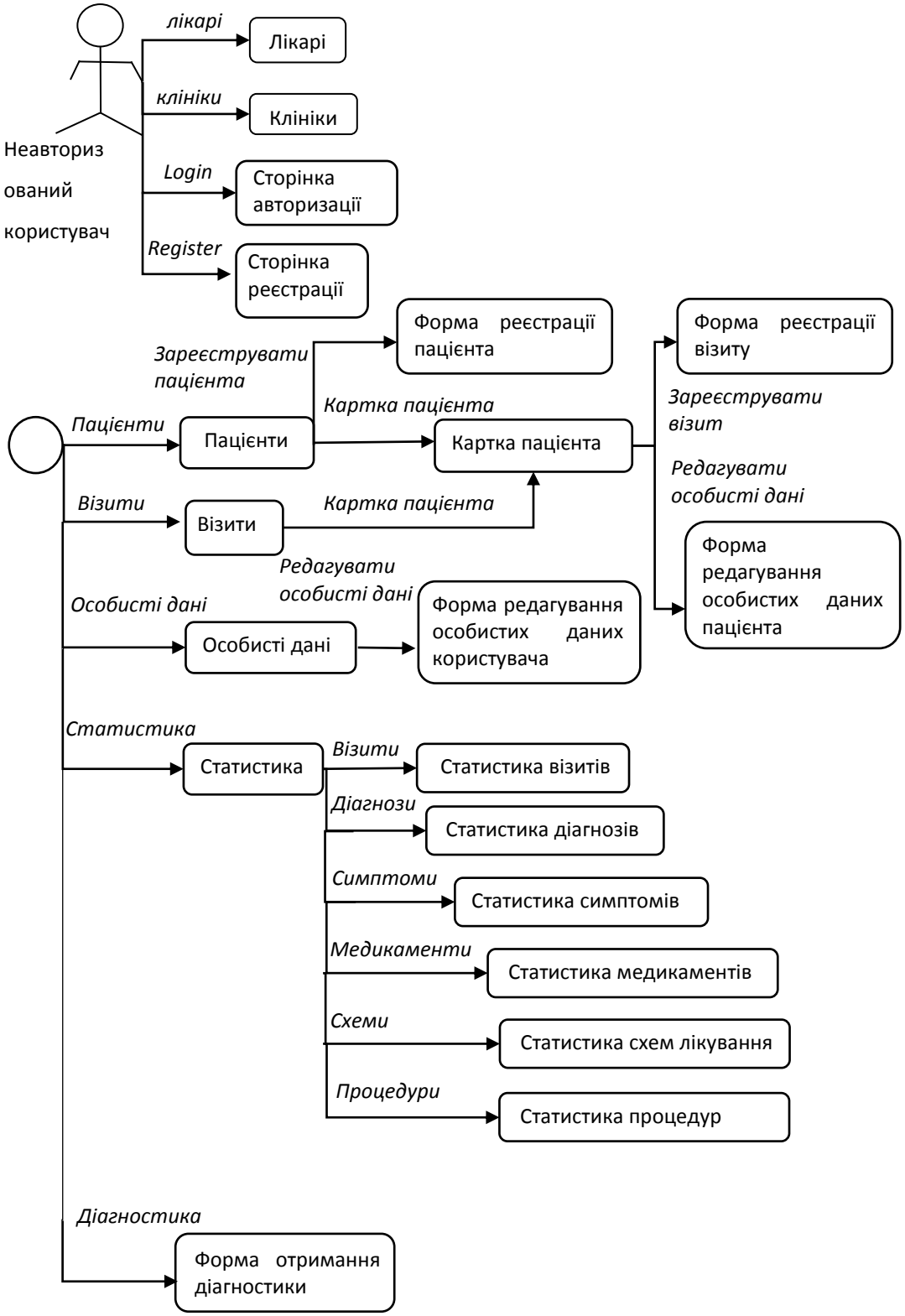
47. Mysql. Головна сторінка. URL: https://www.mysql.com/

48. Habr. SQL запити швидко. URL: https://habr.com/ru/post/480838/

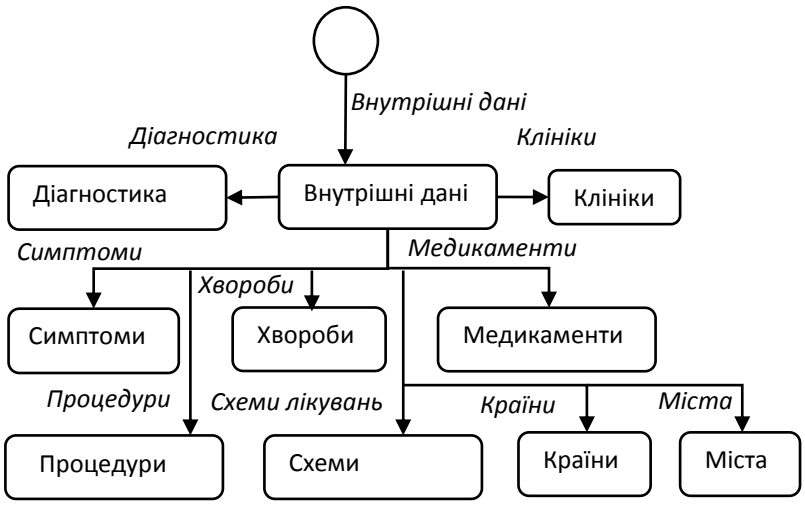
ДОДАТКИ

Додаток А

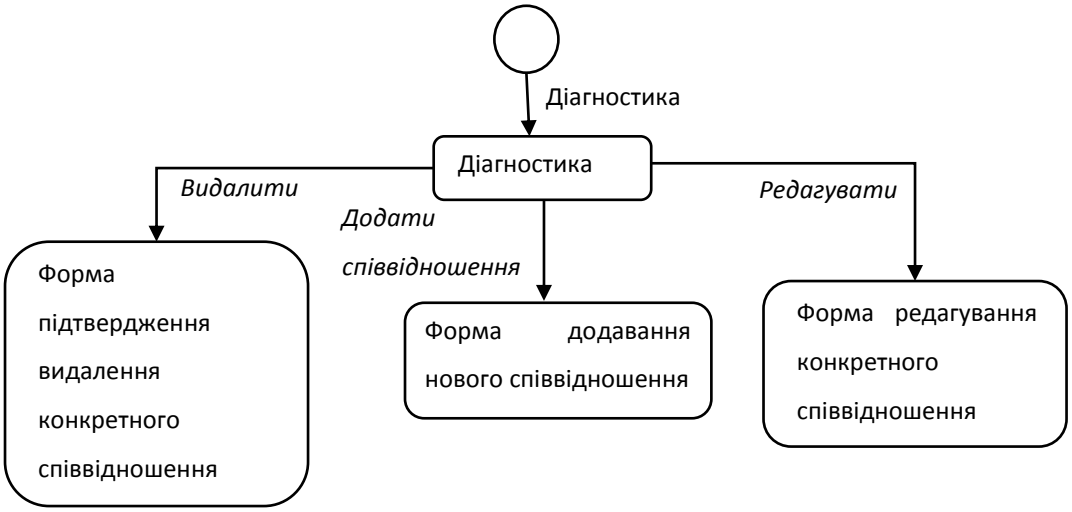
UML діаграма сайту



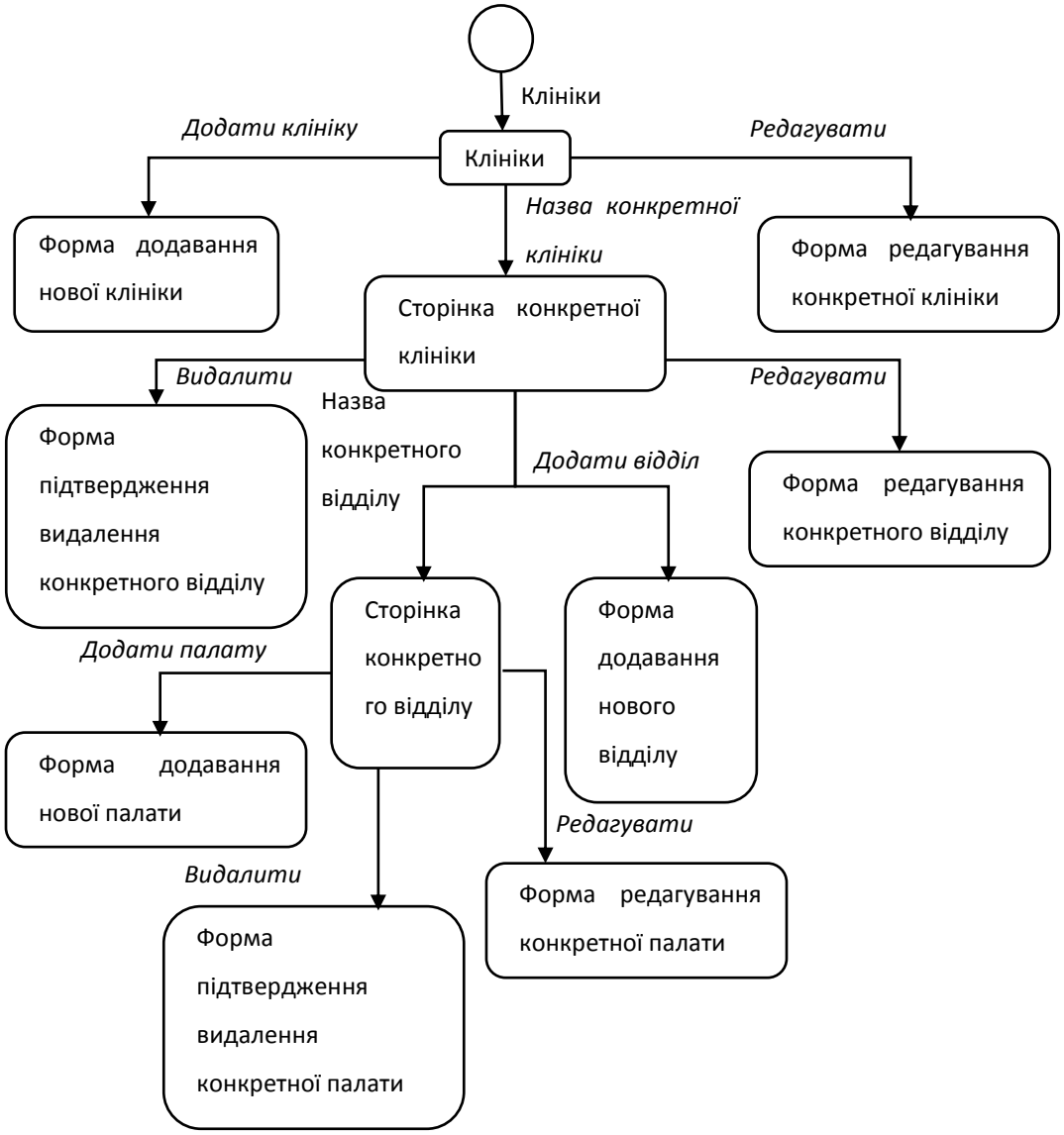
UML-діаграма програми для звичайного користувача



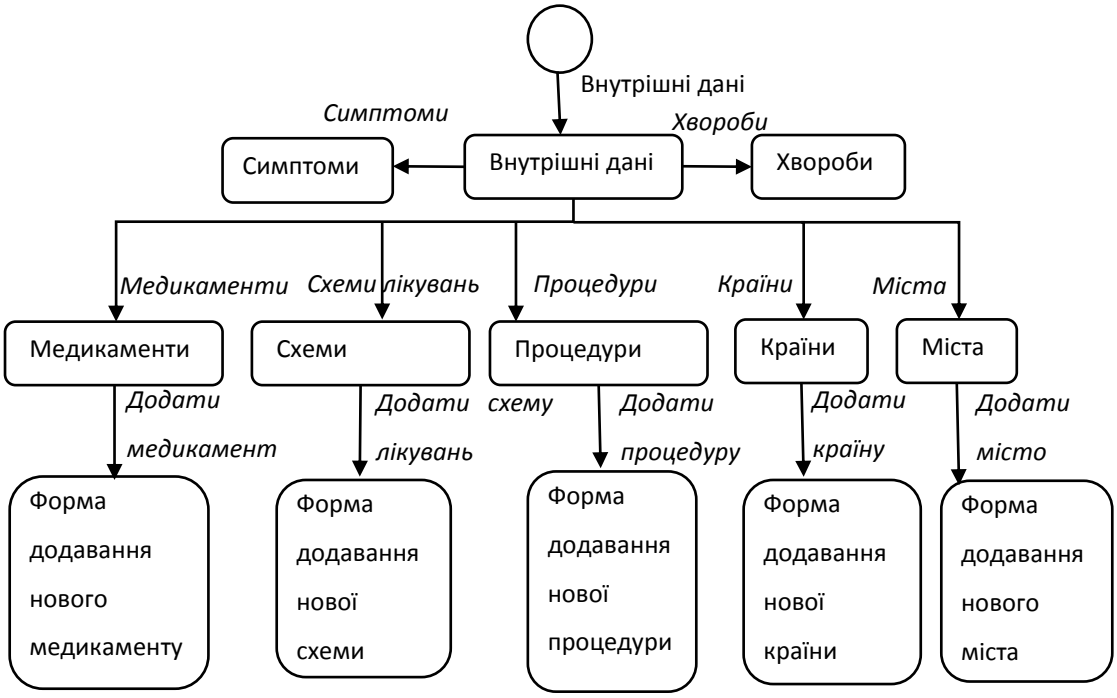
UML-діаграма програми для користувача з правами адміністратора



UML-діаграма програми для адміністратора на сторінці «Діагностика»



UML-діаграма програми для адміністратора на сторінці «Клініки»



UML-діаграма програми для сторінки «Внутрішні дані» відносно сторінок «Симптоми», «Хвороби», «Медикаменти», «Процедури», «Схеми лікувань», «Країни», «Міста».

Додаток Б

Таблиці бази даних автоматизованої системи допомоги в діагностуванні хвороби терапевтом

Таблиця Б.1 – Атрибути таблиці «Presence»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	patient_id	int	Зовнішній ключ до таблиці «Пацієнти»
3	start_at	datetime	Дата початку візиту
4	end_at	datetime	Дата закінчення візиту
3	doctor_id	int	Зовнішній ключ до таблиці «Працівники»
4	created_at	datetime	Дата створення запису
5	updated_at	datetime	Дата обновлення запису

Таблиця Б.2 – Атрибути таблиці «Patients»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	city_id	int	Зовнішній ключ до таблиці «Міста»
3	doctor_id	int	Зовнішній ключ до таблиці «Працівники»
4	last_name	int	Прізвище
5	first_name	int	Ім’я
6	father_name	varchar	По-батькові
7	street	varchar	Вулиця
8	house	varchar	Будинок
9	flat	varchar	Квартира
10	phone_number	varchar	Номер телефону
11	created_at	datetime	Дата створення запису
12	updated_at	datetime	Дата обновлення запису

Таблиця Б.3 – Атрибути таблиці «Users»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	employee_id	int	Зовнішній ключ до таблиці «Працівники»
3	login	varchar	Логін
4	email	varchar	Емейл
6	password	varchar	Пароль
7	role_id	int	Зовнішній ключ до таблиці «Ролі»
8	created_at	datetime	Дата створення запису
9	updated_at	datetime	Дата обновлення запису

Таблиця Б.4 – Атрибути таблиці «Employees»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	city_id	int	Зовнішній ключ до таблиці «Міста»
3	job_id	int	Зовнішній ключ до таблиці «Посади»
4	department_id	int	Зовнішній ключ до таблиці «Відділи»
5	room_id	int	Зовнішній ключ до таблиці «Палати»
6	last_name	varchar	Прізвище
7	first_name	varchar	Ім’я
8	father_name	varchar	По-батькові
9	street	varchar	Вулиця
10	house	varchar	Будинок
11	flat	varchar	Номер квартири
12	phone_number	varchar	Номер телефону
13	about	text	Біографія
14	image	varchar	Посилання на фотографію
15	created_at	datetime	Дата створення запису
16	updated_at	datetime	Дата обновлення запису

Таблиця Б.5 – Атрибути таблиці «Cities»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	country_id	int	Зовнішній ключ до таблиці Країни
3	name	varchar	Назва
4	created_at	datetime	Дата створення запису
5	updated_at	datetime	Дата обновлення запису

Таблиця Б.6 – Атрибути таблиці «Clinics»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	city_id	int	Зовнішній ключ до таблиці Міста
3	name	varchar	Назва
4	street	varchar	Вулиця
5	house	varchar	Будинок
6	phone_number	varchar	Номер телефону
7	type	varchar	Тип
8	schedule	varchar	Розклад
9	image	image	Аватар
10	created_at	datetime	Дата створення запису
11	updated_at	datetime	Дата обновлення запису

Таблиця Б.7 – Атрибути таблиці «Countries»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	name	varchar	Назва
3	created_at	datetime	Дата створення запису
4	updated_at	datetime	Дата обновлення запису

Таблиця Б.8 – Атрибути таблиці «Departments»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	clinic_id	int	Зовнішній ключ до таблиці Клініки
3	name	varchar	Назва
4	created_at	datetime	Дата створення запису
5	updated_at	datetime	Дата обновлення запису

Таблиця Б.9 – Атрибути таблиці «Diseases»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	name	varchar	Назва
3	created_at	datetime	Дата створення запису
4	updated_at	datetime	Дата обновлення запису

Таблиця Б.10 – Атрибути таблиці «Jobs»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	name	varchar	Назва
3	created_at	datetime	Дата створення запису
4	updated_at	datetime	Дата обновлення запису

Таблиця Б.11 – Атрибути таблиці «Medicaments»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	name	varchar	Назва
3	created_at	datetime	Дата створення запису
4	updated_at	datetime	Дата обновлення запису

Таблиця Б.12 – Атрибути таблиці «Procedures»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	name	varchar	Назва
3	created_at	datetime	Дата створення запису
4	updated_at	datetime	Дата обновлення запису

Таблиця Б.13 – Атрибути таблиці «Rooms»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	department_id	int	Зовнішній ключ до таблиці Відділи
3	number	varchar	Номер палати
4	created_at	datetime	Дата створення запису
5	updated_at	datetime	Дата обновлення запису

Таблиця Б.14 – Атрибути таблиці «Symptoms»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	name	int	Назва
3	created_at	datetime	Дата створення запису
4	updated_at	datetime	Дата обновлення запису

Таблиця Б.15 – Атрибути таблиці «Treatments»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	name	varchar	Назва
3	created_at	datetime	Дата створення запису
4	updated_at	datetime	Дата обновлення запису

Таблиця Б.16 – Атрибути таблиці «Presence_Diseases»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	presence_id	int	Зовнішній ключ до таблиці «Patients»
3	disease_id	int	Зовнішній ключ до таблиці «Diseases»
4	description	text	Доповнення
5	date_plan	datetime	Планова дата призначення діагнозу
6	date_fact	datetime	Фактична дата призначення діагнозу
7	created_at	datetime	Дата створення запису
8	updated_at	datetime	Дата обновлення запису

Таблиця Б.17 – Атрибути таблиці «Presence_Procedure»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	presence_id	int	Зовнішній ключ до таблиці «Procedures»
3	procedure_id	int	Зовнішній ключ до таблиці «Employees»
4	description	text	Доповнення
5	date_plan	datetime	Планова дата проведення процедур
6	date_fact	datetime	Фактична дата проведення процедур
7	created_at	datetime	Дата створення запису
8	updated_at	datetime	Дата обновлення запису

Таблиця Б.18 – Атрибути таблиці «Presence_Symptom»

№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	presence_id	int	Зовнішній ключ до таблиці Symptoms
3	symptom_id	int	Зовнішній ключ до таблиці Employees
4	description	varchar	Доповнення
5	date_plan	datetime	Планова дата призначення симптому
6	date_fact	datetime	Фактична дата призначення симптому
7	created_at	datetime	Дата створення запису
8	updated_at	datetime	Дата обновлення запису

Таблиця Б.19 – Атрибути таблиці «Presence_Treatment»

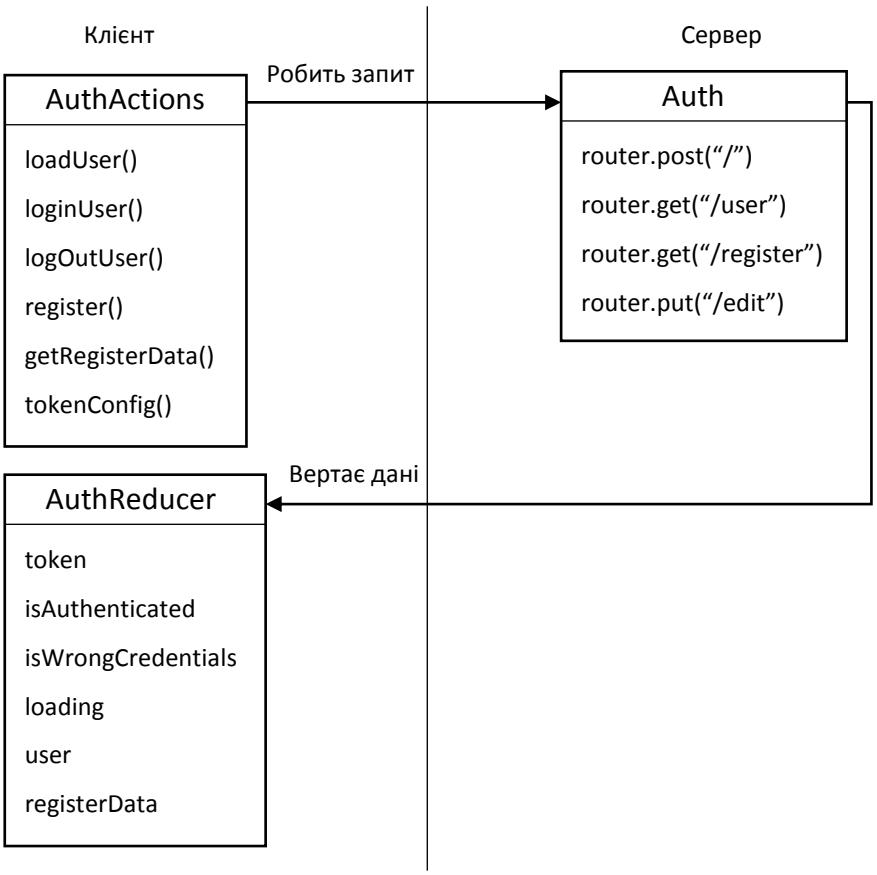
№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	presence_id	int	Зовнішній ключ до таблиці Patients
3	treatment_id	int	Зовнішній ключ до таблиці Treatments
4	description	varchar	Доповнення
5	date_plan	datetime	Планова дата призначення схеми лікування
6	date_fact	datetime	Фактична дата призначення схеми лікування
7	created_at	datetime	Дата створення запису
8	updated_at	datetime	Дата обновлення запису

Таблиця Б.20 – Атрибути таблиці «Symptom_Diseases»

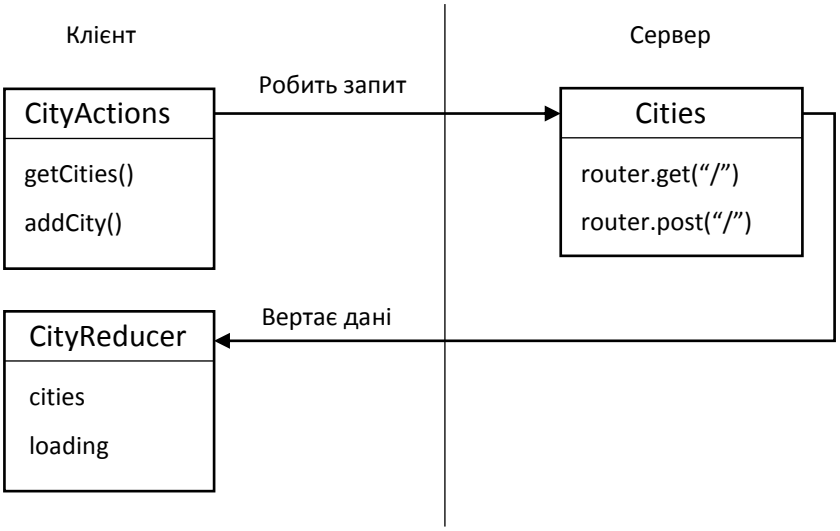
№	Назва атрибуту	Тип даних	Опис
1	id	int	Лічильник, унікальний
2	symptom_id	int	Зовнішній ключ до таблиці Symptoms
3	disease_id	int	Зовнішній ключ до таблиці Diseases
4	created_at	datetime	Дата створення запису
5	updated_at	datetime	Дата обновлення запису

Додаток В

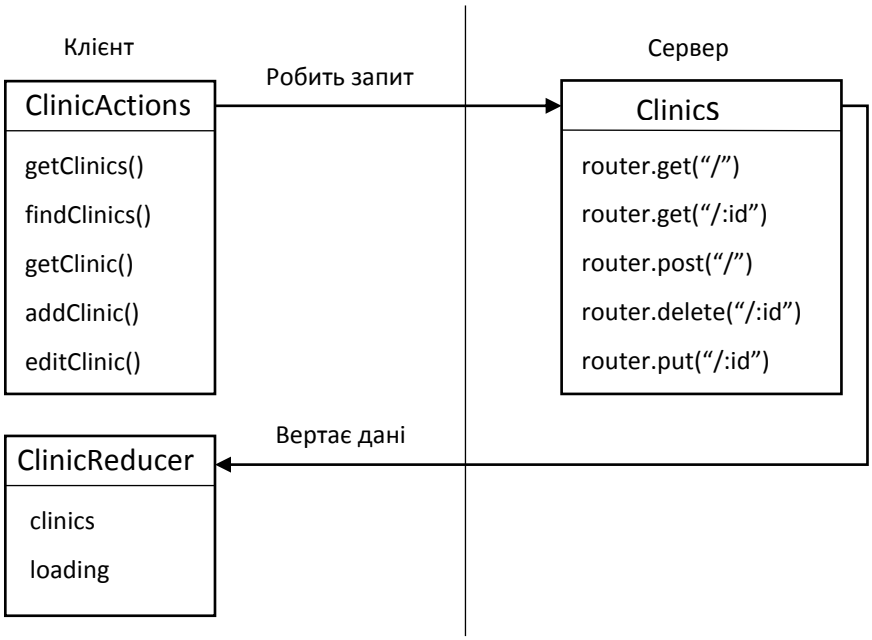
Функціональні блоки програми



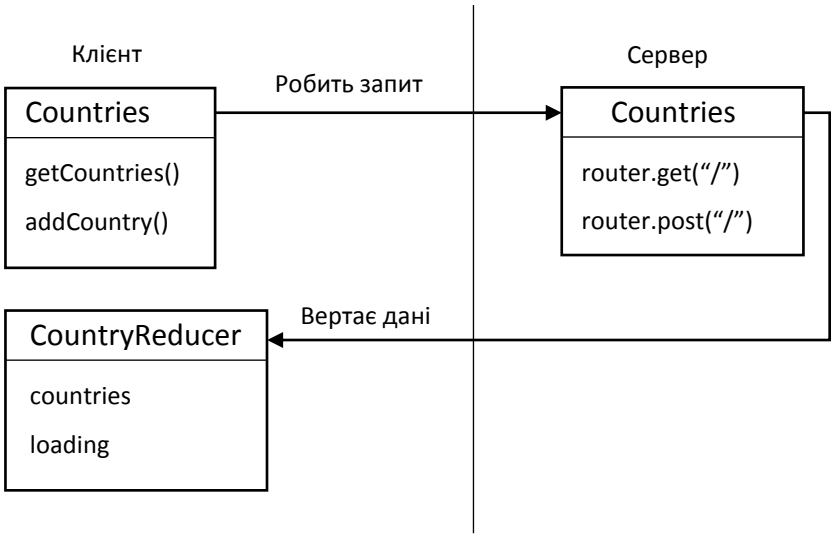
Функціональний блок «Auth»



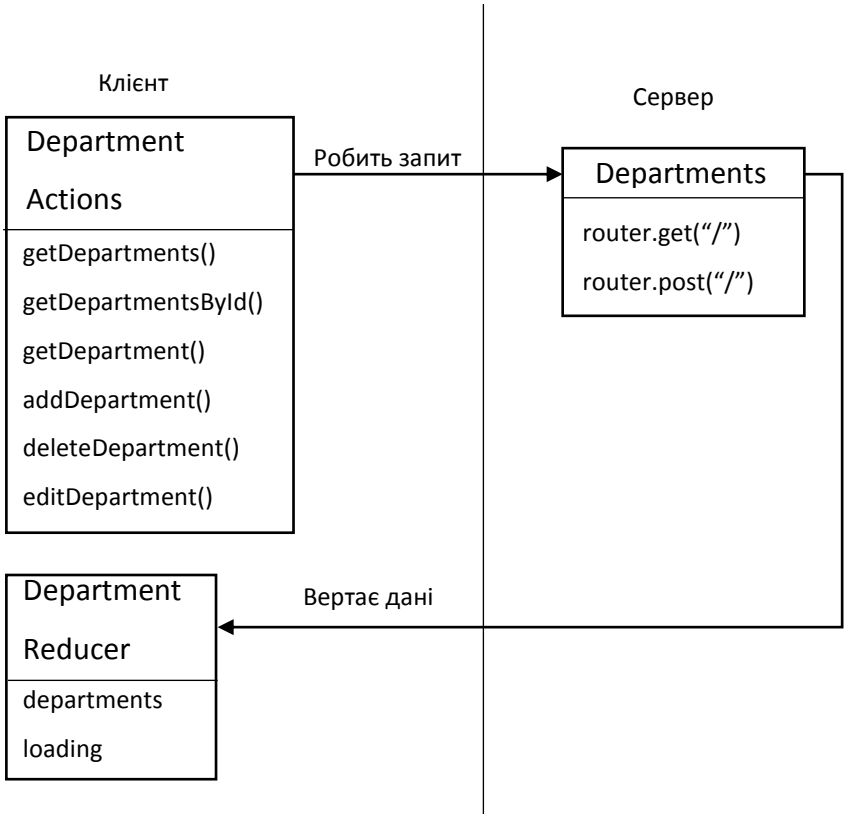
Функціональний блок «Cities»



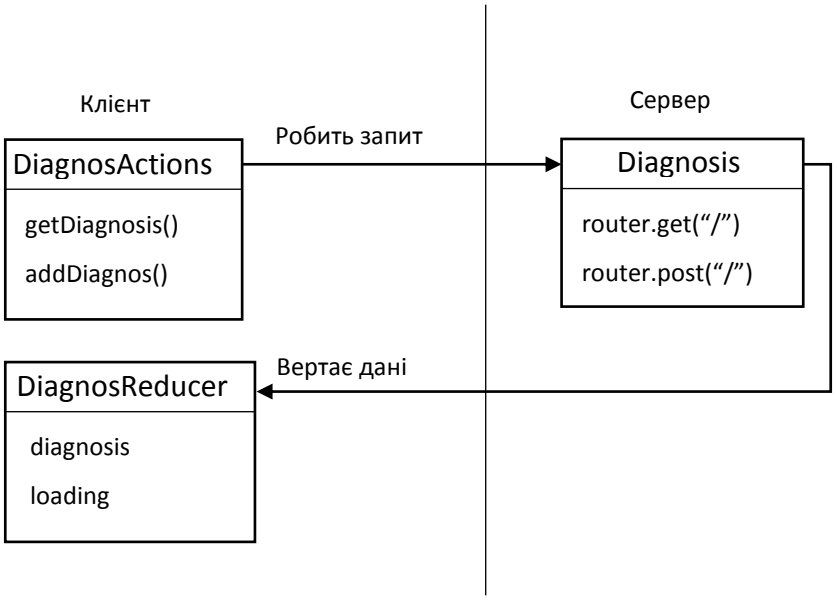
Функціональний блок «Clinics»



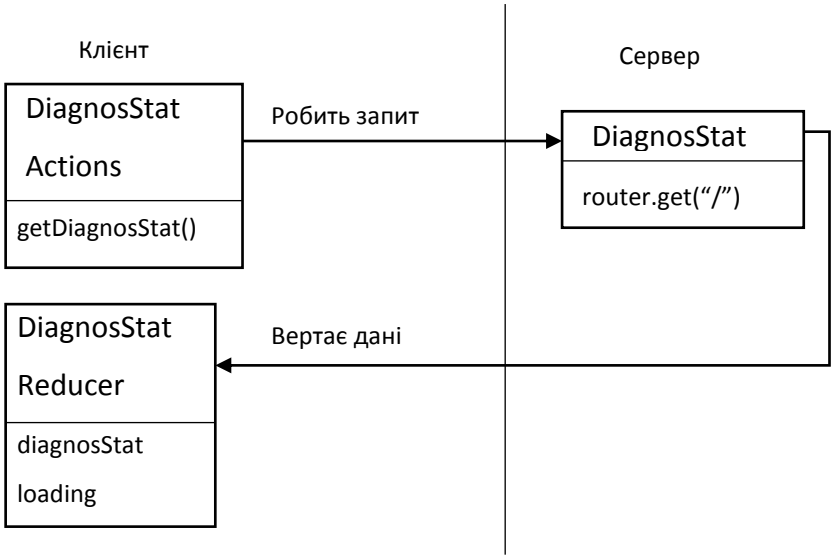
Функціональний блок «Countries»



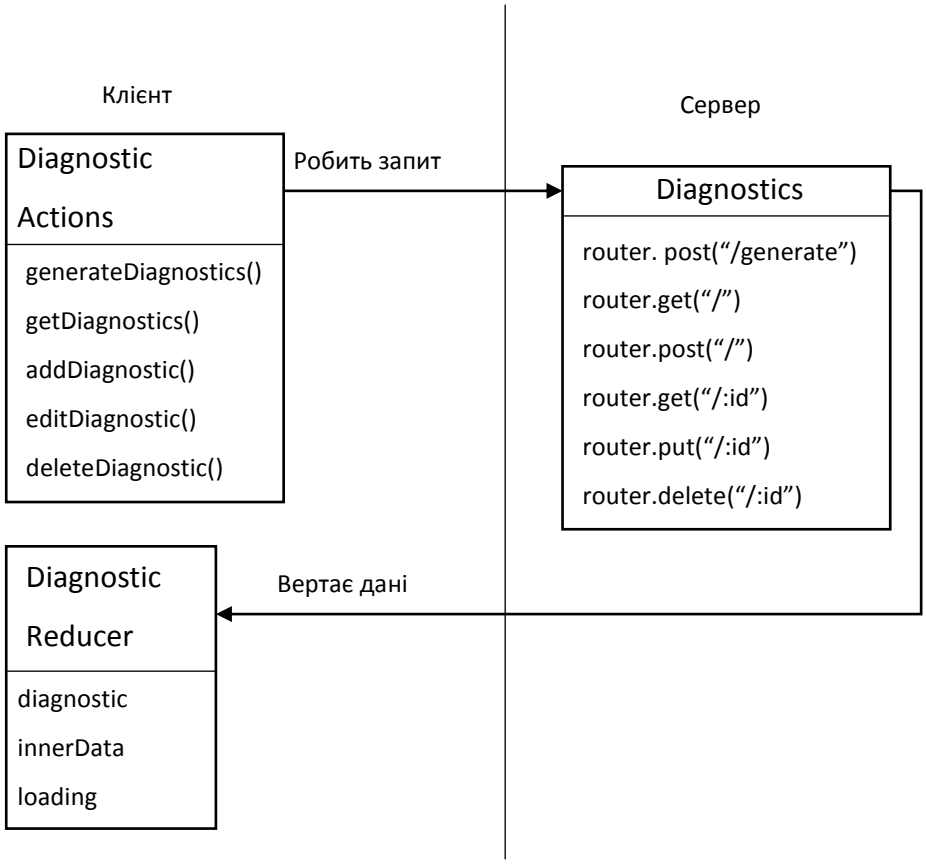
Функціональний блок «Departments»



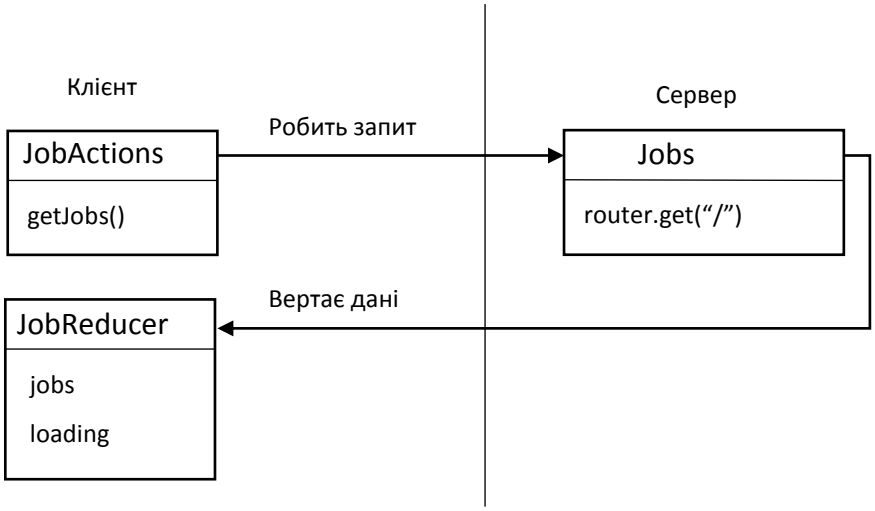
Функціональний блок «Diagnosis»



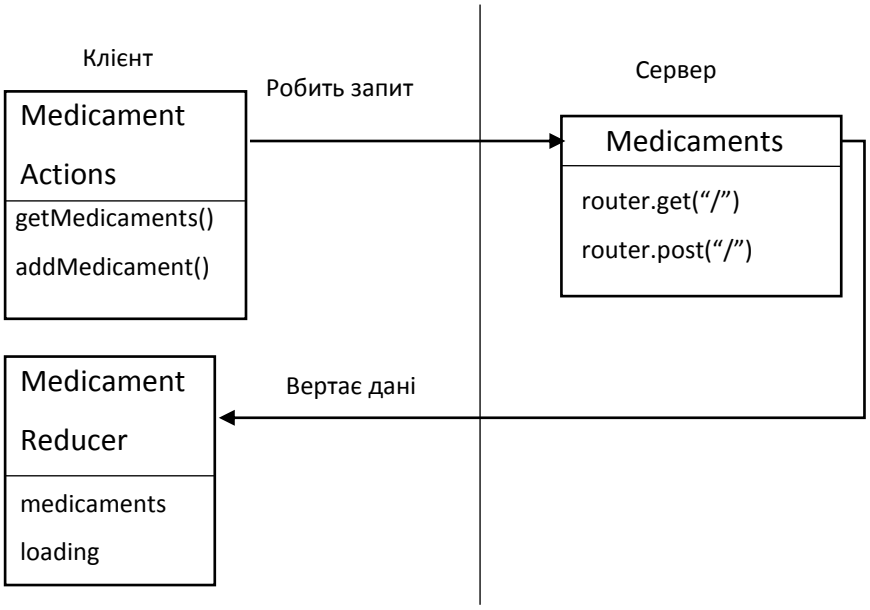
Функціональний блок «DiagnosStat»



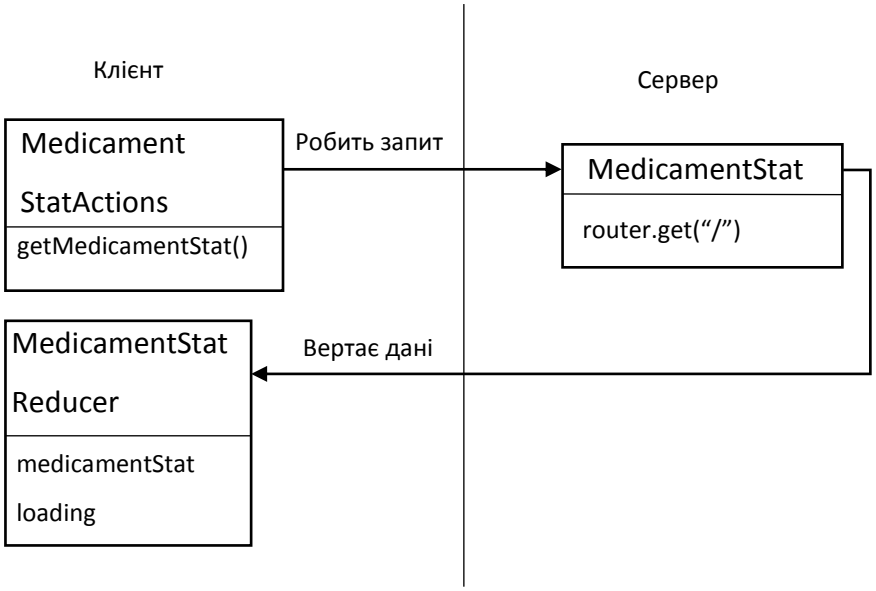
Функціональний блок «Diagnostics»



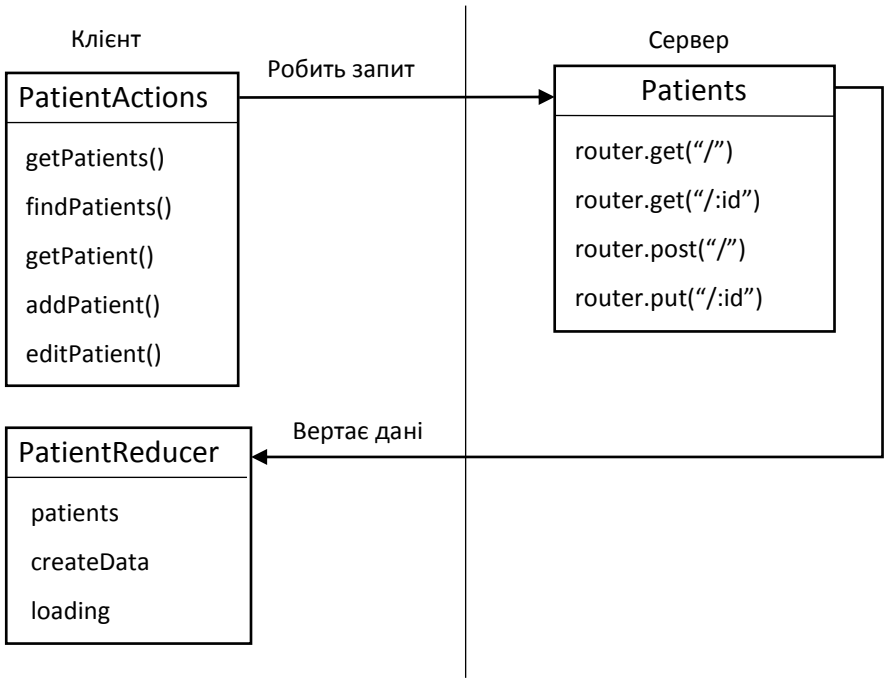
Функціональний блок «Jobs»



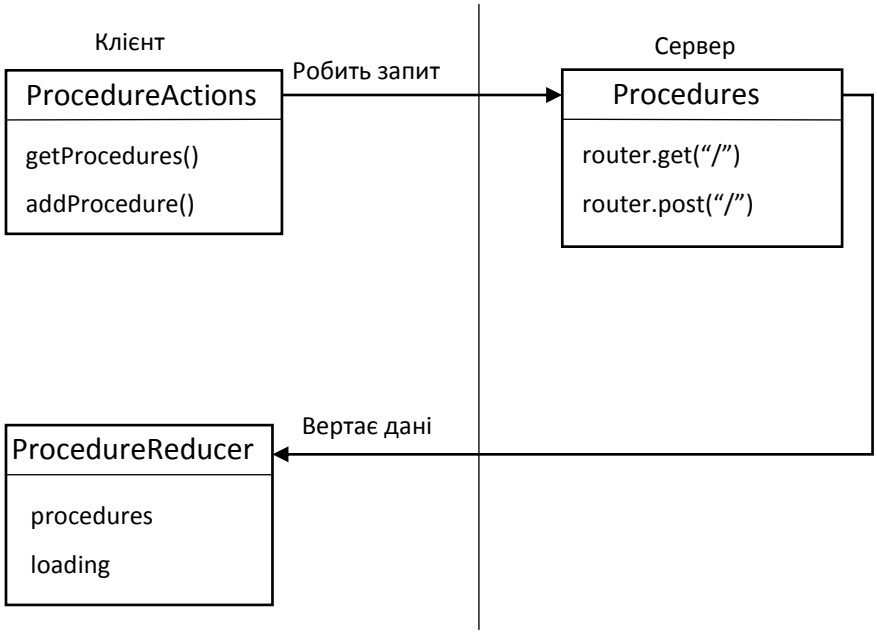
Функціональний блок «Medicaments»



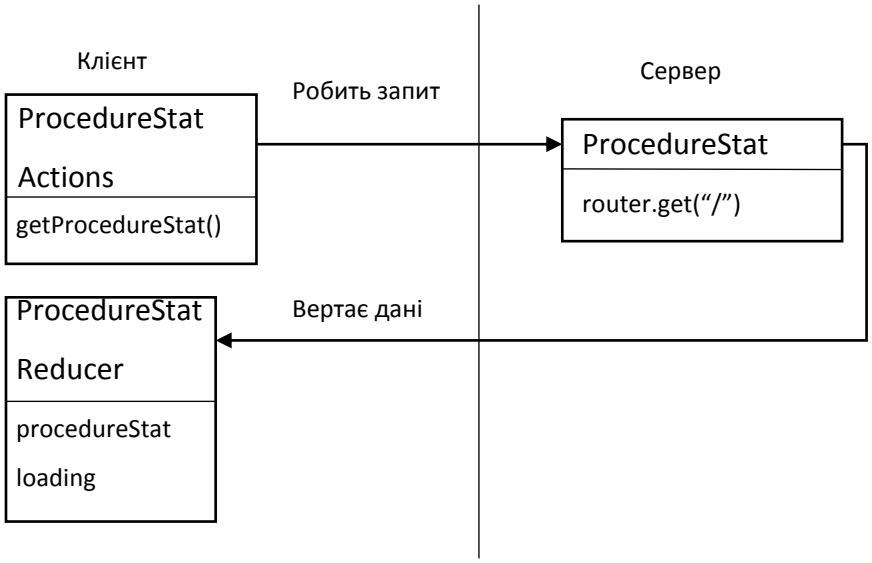
Функціональний блок «MedicamentStat»



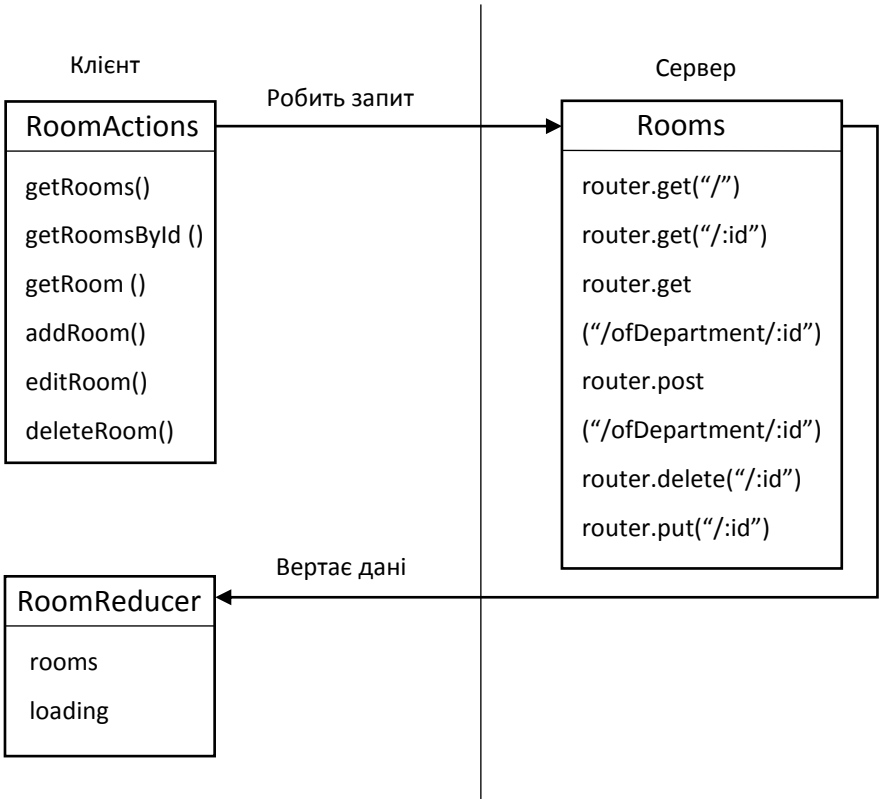
Функціональний блок «Patients»



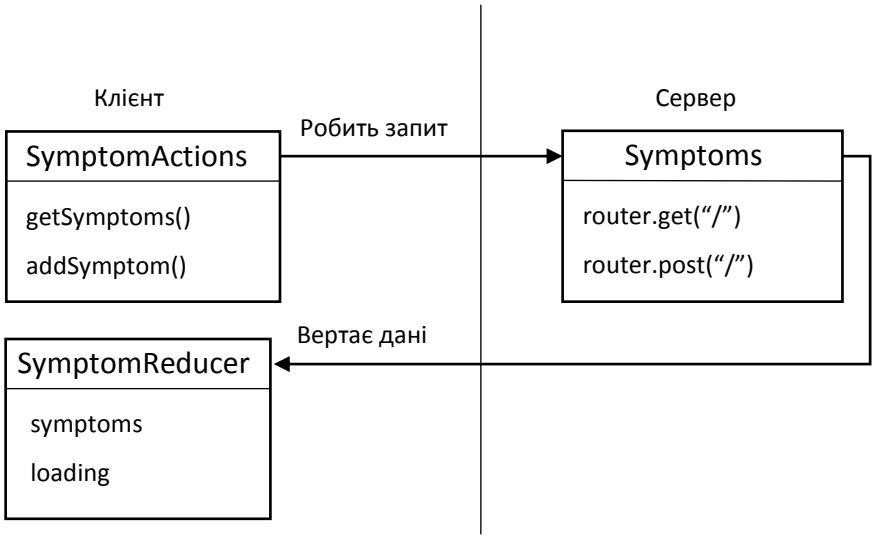
Функціональний блок «Procedures»



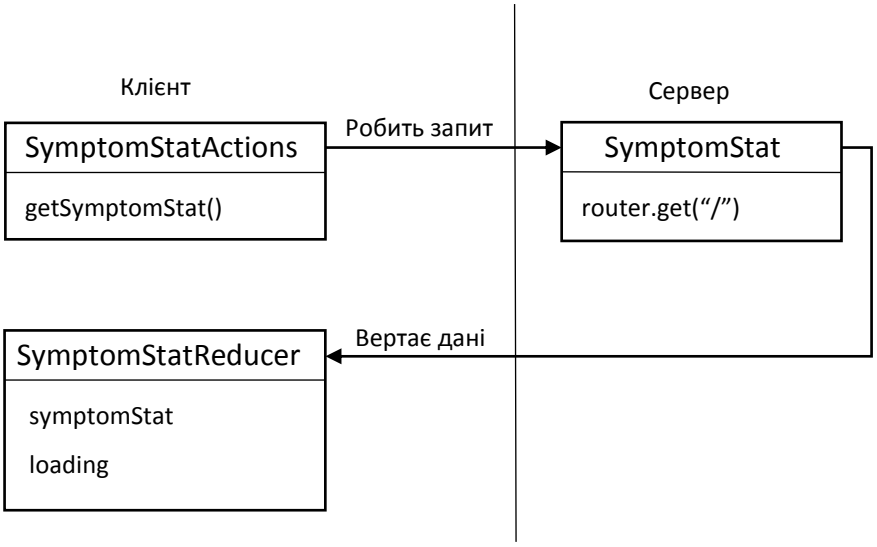
Функціональний блок «ProcedureStat»



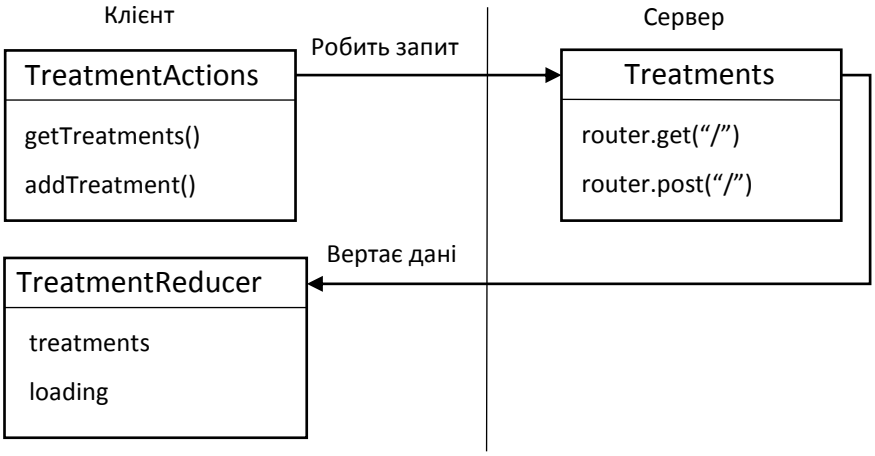
Функціональний блок «Rooms»



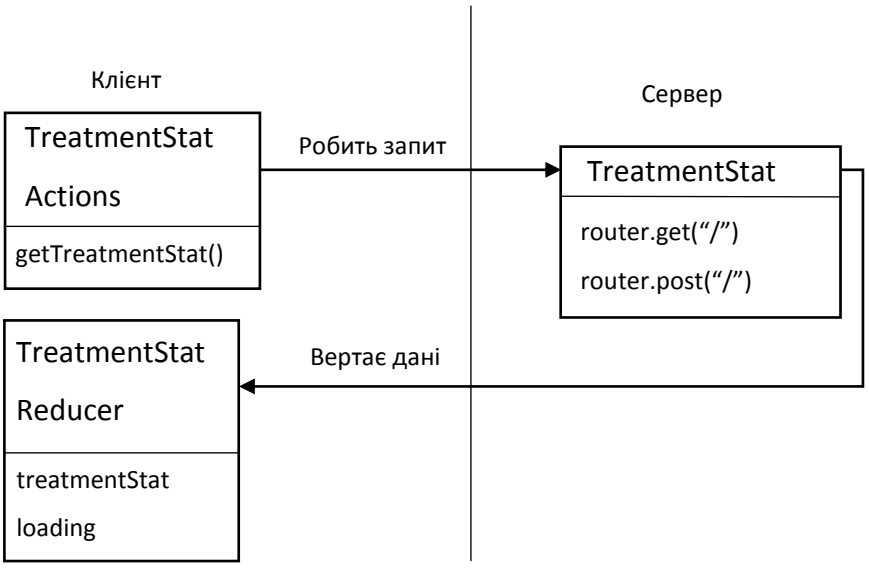
Функціональний блок «Symptoms»



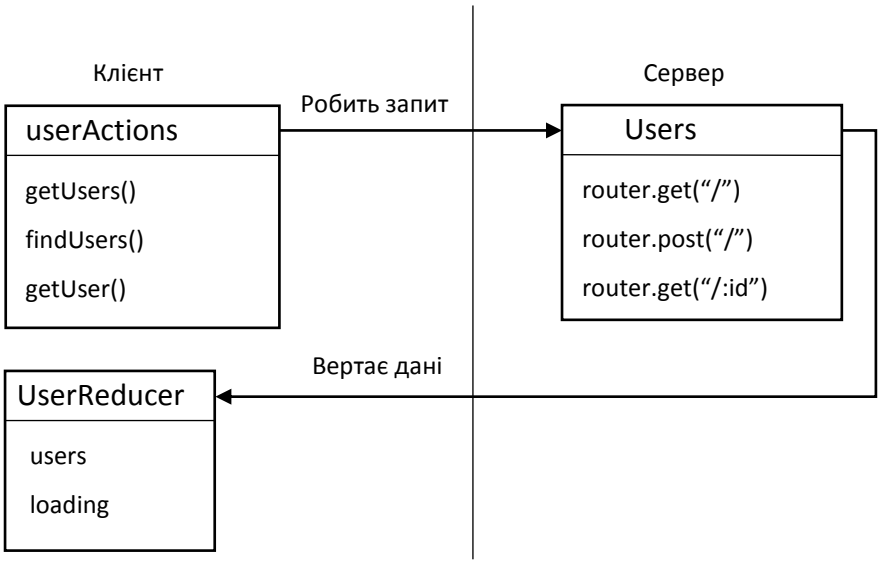
Функціональний блок «SymptomStat»



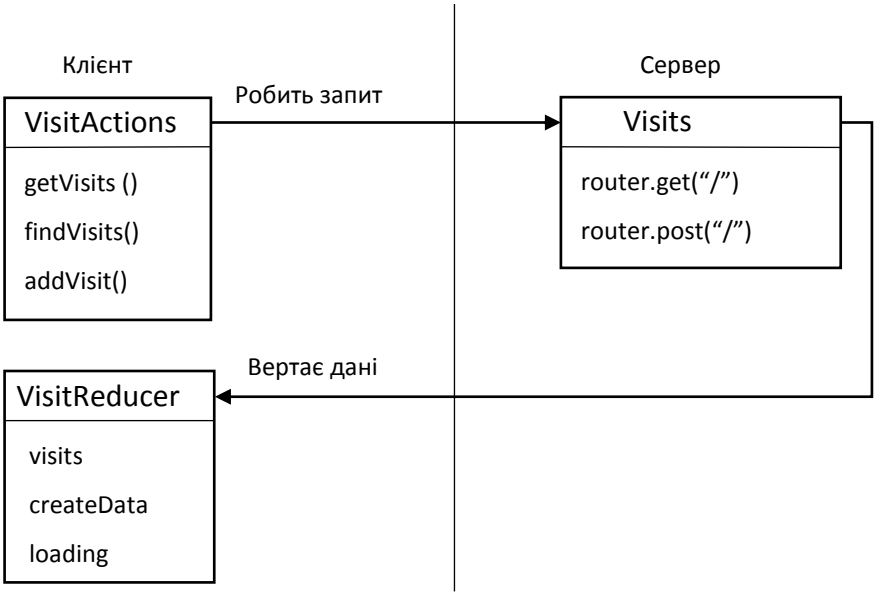
Функціональний блок «Treatments»



Функціональний блок «TreatmentStat»



Функціональний блок «Users»



Функціональний блок «Visits»

Додаток Г

Програмні коди

Лістинг authActions.js

```
import axios from "axios";
import { returnErrors } from "../error/errorActions";
import {
  USER_LOADED,
  USER_LOADING,
  USER_UNLOADED,
  USER_EDIT,
  AUTH_ERROR,
  LOGIN_SUCCESS,
  LOGIN_FAIL,
  LOGOUT_SUCCESS,
  REGISTER_SUCCESS,
  REGISTER_FAIL,
  REGISTER_FORM,
} from "./authTypes";

// Check token and load user
export const loadUser = () => (dispatch, getState) => {
  // User loading
  dispatch({ type: USER_LOADING });

  axios
    .get("/api/auth/user", tokenConfig(getState))
    .then((res) => dispatch({ type: USER_LOADED, payload:
res.data }))
    .catch((err) => {
      dispatch(returnErrors(err.response.data,
err.response.status));
      dispatch({ type: AUTH_ERROR });
    });
};

// Login user
export const loginUser = (data) => (dispatch) => {
  // Headers
  const config = {
    headers: {
      "Content-Type": "application/json",
    },
  };
  // Request body
  const body = JSON.stringify(data);

  axios
    .post("/api/auth", body, config)
    .then((res) => {
      dispatch({
        type: LOGIN_SUCCESS,
        payload: res.data,
      });
      dispatch(loadUser());
    })
    .catch((err) => {
      dispatch(
        returnErrors(err.response.data,    err.response.status,
"LOGIN_FAIL")
      );
      dispatch({
        type: LOGIN_FAIL,
      });
    });
};

// Log out user
export const logOutUser = () => (dispatch) => {
  dispatch({
    type: USER_UNLOADED,
    payload: {},
  });
};

// Edit User
export const editUser = (data) => (dispatch) => {
  const config = {
    headers: {
      "Content-Type": "application/json",
    },
  };
  const body = JSON.stringify(data);
  dispatch({ type: USER_LOADING });
  axios
    .put(`/api/auth/edit`, body, config)
    .then((res) => {
      dispatch({
        type: USER_EDIT,
        payload: res.data,
      });
    })
    .catch((err) => {
      dispatch(
        returnErrors(err.response.data,    err.response.status,
"EDIT_USER ERROR")
      );
    });
};

// Get register form data
export const getRegisterData = () => (dispatch) => {
  axios.get("/api/auth/register").then((res) => {
    dispatch({
      type: REGISTER_FORM,
      payload: res.data,
    });
  });
};

// Register User
export const register = (data) => (dispatch) => {
  // Headers
  const config = {
    headers: {
      "Content-Type": "application/json",
    },
  };

  // Request body
  const body = JSON.stringify(data);

  axios
    .post("/api/users", body, config)
    .then((res) => {
      dispatch({
        type: REGISTER_SUCCESS,
        payload: res.data,
      });
      dispatch(loadUser());
    })
    .catch((err) => {
      dispatch(
        returnErrors(err.response.data,    err.response.status,
"REGISTER_FAIL")
      );
      dispatch({
        type: REGISTER_FAIL,
      });
    });
};

// Setup config/headers and token
export const tokenConfig = (getState) => {
  //Get token from localStorage
  const token = getState().auth.token;

  //Headers
  const config = {
    headers: {
      "Content-type": "application/json",
    },
  };

  // If token add to headers
  if (token) {
    config.headers["x-auth-token"] = token;
  }

  return config;
};
```

Лістинг AuthReducer.js

```
import {
  USER_LOADED,
  USER_LOADING,
  USER_UNLOADED,
  USER_EDIT,
  AUTH_ERROR,
```

```
  LOGIN_SUCCESS,
  LOGIN_FAIL,
  LOGOUT_SUCCESS,
  REGISTER_SUCCESS,
  REGISTER_FAIL,
  REGISTER_FORM,
} from "../actions/auth/authTypes";

const initialState = {
  token: localStorage.getItem("token"),
  isAuthenticated: false,
  isWrongCredentials: false,
  loading: false,
  user: {},
  registerData: [],
};

export default function (state = initialState, action) {
  switch (action.type) {
    case USER_LOADING:
      console.log("user loading");
      return {
        ...state,
        loading: true,
      };
    case USER_LOADED:
      return {
        ...state,
        isAuthenticated: true,
        loading: false,
        user: action.payload,
      };
    case USER_EDIT:
      return {
        ...state,
        user: { ...state.user, ...action.payload },
        loading: false,
      };
    case USER_UNLOADED:
      localStorage.removeItem("token");
      return {
        ...state,
        isAuthenticated: false,
        user: {},
      };
    case LOGIN_SUCCESS:
    case REGISTER_SUCCESS:
      localStorage.setItem("token", action.payload.token);
      return {
        ...state,
        ...action.payload,
        isAuthenticated: true,
        loading: false,
        isWrongCredentials: false,
      };
    case LOGIN_FAIL:
      localStorage.removeItem("token");
      return {
        ...state,
        token: null,
        user: null,
        isAuthenticated: false,
        loading: false,
        isWrongCredentials: true,
      };
    case AUTH_ERROR:
    case LOGOUT_SUCCESS:
    case REGISTER_FAIL:
      localStorage.removeItem("token");
      return {
        ...state,
        token: null,
        user: null,
        isAuthenticated: false,
        loading: false,
      };
    case REGISTER_FORM:
      return {
        ...state,
        registerData: action.payload,
      };
    default:
      return state;
  }
}

const express = require("express");
const router = express.Router();
const bcrypt = require("bcryptjs");
const config = require("config");
const jwt = require("jsonwebtoken");
const auth = require("../../middleware/auth");
const conn = require("../../config/db");
```

Лістинг auth.js

```
// @route POST /api/auth
// @desc Auth user
// @access public
router.post("/", (req, res) => {
  const { email, password } = req.body;

  if (!email || !password) {
    return res.status(400).json({ msg: "Please enter all
fields" });
  }

  conn.query(
    "SELECT * FROM users WHERE " + `users.email =
\"${email}\";`,
    (err, results, fields) => {
      if (err) return res.json(err);
      if (results.length === 0)
        return res.status(400).json({ msg: "User does not
exists" });
      const result = results[0];
      bcrypt
        .compare(password, result.password)
        .then((isMatch) => {
          if (!isMatch)
            return res.status(400).json({ msg: "Invalid
credentials" });
          jwt.sign(
            { id: result.id },
            config.get("jwtSecret"),
            { expiresIn: 36000 },
            (err, token) => {
              if (err) throw err;
              return res.json({
                token,
                user: {
                  id: result.id,
                  login: result.login,
                  email: result.email,
                },
              });
            }
          );
        })
        .catch((err) => res.status(400).json({ msg: "Password
error" }));
    }
  );
});

// @route GET /api/auth/user
// @desc Get user data
// @access private
router.get("/user", auth, (req, res) => {
  conn.query(
    "SELECT u.login, u.email, u.role_id, u.id, c.name as city,
co.name as country, " +
      "d.name as department, e.about, e.last_name as lastName,
" +
      "e.first_name as firstName, e.father_name as fatherName,
" +
      "e.street, e.flat, e.house, e.image as image, " +
      "e.phone_number as phoneNumber, j.name as job " +
      "FROM users u INNER JOIN employees e ON u.employee_id =
e.id " +
      "INNER JOIN cities c ON e.city_id = c.id " +
      "INNER JOIN countries co ON c.country_id = co.id " +
      "INNER JOIN departments d ON e.department_id = d.id " +
      "INNER JOIN jobs j on e.job_id = j.id " +
      `WHERE u.id = ${req.user.id};`,
    (err, results, fields) => {
      if (err) return res.json(err);
      return res.json(results[0]);
    }
  );
});

// @route GET /api/auth/register
// @desc Get register form data
// @access public
router.get("/register", (req, res) => {
  conn.query("SELECT email FROM users", (err, results, fields)
=> {
    if (err) return res.status(400).json(err);
    return res.json(results);
```

```
  });
});

// @route PUT /api/users/id
router.put("/edit", (req, res) => {
  const data = req.body;
  if (!{ ...data }) {
    return res.status(400).json({ msg: "Please enter all
fields" });
  }

  conn.query(
    "SELECT id, employee_id FROM users " + `WHERE id =
${data.id}`,
    (err, results, fields) => {
      if (err) return res.status(400).json(err);
      const { id, employee_id } = results[0];
      conn.query(
        `UPDATE users SET login = "${data.login}" ` +
          `WHERE id = ${id}; UPDATE employees SET ` +
          `last_name = "${data.lastName}", first_name =
"${data.firstName}", ` +
          `father_name = "${data.fatherName}", about =
"${data.about}", ` +
          `street = "${data.street}", house = "${data.house}",
` +
          `image = "${data.image}", flat = "${data.flat}", ` +
          `city_id = ${data.cityId}, job_id = ${data.jobId}, `
+
          `department_id = ${data.departmentId} WHERE id =
${employee_id};`,
        (err, results, fields) => {
          if (err) return res.status(400).json(err);
          return res.json(data);
        }
      );
    }
  );
});

module.exports = router;
```

Лістинг Login.js

```
import React from "react";
import { connect } from "react-redux";
import PropTypes from "prop-types";
import { loginUser } from "../../actions/auth/authActions";
import { Redirect, Link } from "react-router-dom";

class Login extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      email: "",
      password: "",
      rememberMe: false,
    };
  }

  onSubmit = (e) => {
    const { email, password } = this.state;
    e.preventDefault();
    this.props.loginUser({ email, password });
  };

  onBaseInputChange = (e) => {
    this.setState({
      ...this.state,
      [e.target.name]: e.target.value,
    });
  };

  redirect = () => {
    const { isAuthenticated } = this.props.auth;
    if (isAuthenticated) {
      return <Redirect to="/" />;
    }
  };

  setErrorClass = () => {
    const { isWrongCredentials } = this.props.auth;
    if (isWrongCredentials) {
      return "is-invalid";
    } else {
      return "";
    }
  };

  setErrorMsg = () => {
    const { isWrongCredentials } = this.props.auth;
    if (isWrongCredentials) {
      return (
        <span className="invalid-feedback" role="alert">
          <strong>Неправильний логін або пароль</strong>
        </span>
      );
    } else {
      return <span></span>;
    }
  };

  render() {
    return (
      <div className="container">
        <div className="row justify-content-center">
          <div className="col-md-8">
            <div className="card">
              <div className="card-header">Login</div>

              <div className="card-body">
                <form method="POST" onSubmit={this.onSubmit}>
                  <div className="form-group row">
                    <label
                      htmlFor="email"
                      className="col-md-4 col-form-label text-
md-right"
                    >
                      E-Mail Address
                    </label>

                    <div className="col-md-6">
                      <input
                        id="email"
                        type="email"
                        className={`form-control
${this.setErrorClass()}`}
                        name="email"
                        required
                        autoComplete="email"
                        autoFocus
                        onChange={this.onBaseInputChange}
                        value={this.state.email}
                      />

                      {this.setErrorMsg()}
                    </div>
                  </div>

                  <div className="form-group row">
                    <label
                      htmlFor="password"
                      className="col-md-4 col-form-label text-
md-right"
                    >
                      Password
                    </label>

                    <div className="col-md-6">
                      <input
                        id="password"
                        type="password"
                        className="form-control"
                        name="password"
                        required
                        autoComplete="current-password"
                        onChange={this.onBaseInputChange}
                        value={this.state.password}
                      />
                    </div>
                  </div>

                  <div className="form-group row mb-0">
                    <div className="col-md-8 offset-md-4">
                      <button type="submit" className="btn
btn-primary mr-2">
                        Login
                      </button>
                      <Link to="/" className="btn btn-danger"
role="button">
                          Cancel
                        </Link>
                      </div>
                    </div>

                    {this.redirect()}
                  </form>
                </div>
              </div>
```

```
                    </div>
                  </div>
                </div>
              );
            }
          }

          Login.propTypes = {
            loginUser: PropTypes.func.isRequired,
          };

          const mapStateToProps = (state) => ({
            auth: state.auth,
          });

export default connect(mapStateToProps, { loginUser })(Login);
```

Лістинг Register.js

```
import React from "react";
import { connect } from "react-redux";
import { register } from "../../actions/auth/authActions";
import { getCities } from "../../actions/city/cityActions";
import          {          getCountries          }          from
"../../actions/country/countryActions";
import          {          getClinics          }          from
"../../actions/clinic/clinicActions";
import { getJobs } from "../../actions/job/jobActions";
import { getRooms } from "../../actions/room/roomActions";
import         {         getDepartments         }         from
"../../actions/department/departmentActions";
import         {         getRegisterData         }         from
"../../actions/auth/authActions";
import AwsClass from "../../aws/awsApi";
import { getImgBuffer } from "../../aws/imgBuffer";
import PropTypes from "prop-types";
import { Redirect, Link } from "react-router-dom";

class Register extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      user: {
        login: "",
        email: "",
        password: "",
        passwordConfirm: "",
        lastName: "",
        firstName: "",
        fatherName: "",
        about: "",
        city: "",
        country: "",
        street: "",
        house: "",
        flat: "",
        phoneNumber: "",
        clinic: "",
        job: "",
        department: "",
        room: "",
        image: "",
      },
      countries: [],
      cities: [],
      clinics: [],
      jobs: [],
      departments: [],
      emails: [],
      rooms: [],
      errors: { isEmailError: false, isPasswordConfirmError:
false },
      isOverallError: false,
      emailErrorMsg: "Користувач з таким email уже існує",
      passwordConfimMsg: "Паролі не співпадають",
      overallErrorMsg: "Помилка реєстрації, перевірте, будь
ласка, усі поля.",
      imageFile: {},
    };
  }

  onBaseInputChange = (e) => {
    this.setState({
      ...this.state,
      user: {
        ...this.state.user,
        [e.target.name]: e.target.value,
      },
    });
  };

  onPasswordConfirmChange = (e) => {
    const passwordConfirm = e.target.value;
    const { user } = this.state;
    if (passwordConfirm !== user.password) {
      this.setState({
        ...this.state,
        errors:            {            ...this.state.errors,
isPasswordConfirmError: true },
        isOverallError: true,
        user: {
          ...this.state.user,
          [e.target.name]: e.target.value,
        },
      });
    } else {
      this.setState({
        ...this.state,
        errors:            {            ...this.state.errors,
isPasswordConfirmError: false },
        isOverallError: false,
        user: {
          ...this.state.user,
          [e.target.name]: e.target.value,
        },
      });
    }
  };

  setErrorClass = (error) => {
    if (error) {
      return "is-invalid";
    } else {
      return "";
    }
  };

  setErrorMsg = (error, msg) => {
    if (error) {
      return (
        <span className="invalid-feedback" role="alert">
          <strong>{msg}</strong>
        </span>
      );
    } else {
      return <span></span>;
    }
  };

  setOverralErrorMsg = (error, msg) => {
    if (error) {
      return (
        <span     className="text-danger     mx-auto     mb-4"
role="alert">
          <strong>{msg}</strong>
        </span>
      );
    } else {
      return <span></span>;
    }
  };

  onEmailChange = (e) => {
    const email = e.target.value;
    if (this.state.emails.includes(email)) {
      this.setState({
        ...this.state,
        errors: { ...this.state.errors, isEmailError: true },
        isOverallError: true,
        user: {
          ...this.state.user,
          [e.target.name]: e.target.value,
        },
      });
    } else {
      this.setState({
        ...this.state,
        errors: { ...this.state.errors, isEmailError: false },
        isOverallError: false,
        user: {
          ...this.state.user,
          [e.target.name]: e.target.value,
        },
      });
            }
          };

          onImageChange = (e) => {
            this.setState({
```

```
    ...this.state,
    imageFile: e.target.files[0],
  });
};

onCountryChange = (e) => {
  const { cities } = this.props.city;
  const { clinics } = this.props.clinic;
  const { departments } = this.props.department;
  const { rooms } = this.props.room;
  let countryId;
  try {
    countryId = parseInt(e.target.value);
  } catch (err) {
    console.log(err);
  }
  const updatedCities = cities.filter(
    (city) => city.country_id === countryId
  );
  let updatedClinics = [];
  let updatedDepartments = [];
  let updatedRooms = [];
  if (updatedCities.length > 0) {
    updatedClinics = clinics.filter(
      (clinic) => clinic.city_id === updatedCities[0].id
    );
  }
  if (updatedClinics.length > 0) {
    updatedDepartments = departments.filter(
      (d) => d.clinic_id === updatedClinics[0].clinic_id
    );
  }
  if (updatedDepartments.length > 0) {
    updatedRooms = rooms.filter(
      (r) => r.department_id === updatedDepartments[0].id
    );
  }
  this.setState({
    ...this.state,
    cities: updatedCities,
    clinics: updatedClinics,
    departments: updatedDepartments,
    rooms: updatedRooms,
    user: {
      ...this.state.user,
      country: countryId,
    },
  });
};

onCityChange = (e) => {
  const { clinics } = this.props.clinic;
  const { departments } = this.props.department;
  const { rooms } = this.props.room;
  let cityId;
  try {
    cityId = parseInt(e.target.value);
  } catch (err) {
    console.log(err);
  }
  const updatedClinics = clinics.filter(
    (clinic) => clinic.city_id === cityId
  );
  let updatedDepartments = [];
  let updatedRooms = [];
  if (updatedClinics.length > 0) {
    updatedDepartments = departments.filter(
      (d) => d.clinic_id === updatedClinics[0].clinic_id
    );
  }
  if (updatedDepartments.length > 0) {
    updatedRooms = rooms.filter(
      (r) => r.department_id === updatedDepartments[0].id
    );
  }
  this.setState({
    ...this.state,
    clinics: updatedClinics,
    departments: updatedDepartments,
    rooms: updatedRooms,
    user: {
      ...this.state.user,
      city: cityId,
    },
  });
          };

          onClinicChange = (e) => {
            const { departments } = this.props.department;
            const { rooms } = this.props.room;
            let clinicId;
            try {
              clinicId = parseInt(e.target.value);
            } catch (err) {
              console.log(err);
            }
            const updatedDepartments = departments.filter(
              (d) => d.clinic_id === clinicId
            );
            let updatedRooms = [];
            if (updatedDepartments.length > 0) {
              updatedRooms = rooms.filter(
                (r) => r.department_id === updatedDepartments[0].id
              );
            }
            this.setState({
              ...this.state,
              departments: updatedDepartments,
              rooms: updatedRooms,
              user: {
                ...this.state.user,
                clinic: clinicId,
              },
            });
          };

          onDepartmentChange = (e) => {
            const { rooms } = this.props.room;
            let departmentId;
            try {
              departmentId = parseInt(e.target.value);
            } catch (err) {
              console.log(err);
            }
            const updatedRooms = rooms.filter((r) => r.department_id
=== departmentId);
            this.setState({
              ...this.state,
              rooms: updatedRooms,
              user: {
                ...this.state.user,
                department: departmentId,
              },
            });
          };

          onRoomChange = (e) => {
            let roomId;
            try {
              roomId = parseInt(e.target.value);
            } catch (err) {
              console.log(err);
            }
            this.setState({
              ...this.state,
              user: {
                ...this.state.user,
                room: roomId,
              },
            });
          };

          redirect = () => {
            const { isAuthenticated } = this.props.auth;
            if (isAuthenticated) {
              return <Redirect to="/" />;
            }
          };

          onSubmit = (e) => {
            const { user, imageFile } = this.state;
            const { errors } = this.state;
            e.preventDefault();
            if (this.state.isOverallError) {
              alert("Помилка реєстрації");
            } else {
              const awsObject = AwsClass.build().then((aws) => {
                const reader = new FileReader();
                reader.onloadend = () => {
                  const base64 = reader.result;
                  const buffer = getImgBuffer(base64);
                  aws
                    .uploadImage(imageFile.name,  buffer,  user.email,
"doctors")
                    .then((res) =>
                      this.setState(
                        {
                          ...this.state,
                          user: {
                            ...this.state.user,
```

```
                  image: res,
                },
              },
              () => {
                this.props.register(this.state.user);
              }
            )
          );
      };
      reader.readAsDataURL(imageFile);
    });
  }
};

  componentDidMount() {
    this.props.getCities();
    this.props.getCountries();
    this.props.getClinics();
    this.props.getJobs();
    this.props.getDepartments();
    this.props.getRooms();
    this.props.getRegisterData();
  }

  componentDidUpdate(prevProps) {
    if (prevProps !== this.props) {
      const { cities, loading } = this.props.city;
      const { countries } = this.props.country;
      const { clinics } = this.props.clinic;
      const { jobs } = this.props.job;
      const { departments } = this.props.department;
      const { rooms } = this.props.room;
      const { registerData } = this.props.auth;
      const emails = registerData.map((obj) => obj.email);
      if (
        rooms.length > 0 &&
        departments.length > 0 &&
        jobs.length > 0 &&
        clinics.length > 0 &&
        countries.length > 0 &&
        cities.length > 0
      ) {
        const defaultCities = cities.filter(
          (city) => city.country_id === countries[0].id
        );
        const defaultClinics = clinics.filter(
          (clinic) => clinic.city_id === defaultCities[0].id
        );
        const defaultDepartments = departments.filter(
          (department)     =>     department.clinic_id     ===
defaultClinics[0].clinic_id
        );
        const defaultRooms = rooms.filter(
          (r) => r.department_id === defaultDepartments[0].id
        );
        this.setState({
          ...this.state,
          countries: countries,
          cities: defaultCities,
          clinics: defaultClinics,
          jobs: jobs,
          emails: emails,
          departments: defaultDepartments,
          rooms: defaultRooms,
          user: {
            ...this.state.user,
            country: countries[0].id,
            city: defaultCities[0].id,
            clinic: defaultClinics[0].clinic_id,
            job: jobs[0].id,
            department: defaultDepartments[0].id,
            room: defaultRooms[0].id,
          },
        });
      }
    }
  }

  render() {
    const { cities, countries, clinics, jobs, departments,
rooms } = this.state;
                const { isEmailError, isPasswordConfirmError }
= this.state.errors;
                const {
                  emailErrorMsg,
                  passwordConfimMsg,
                  isOverallError,
                  overallErrorMsg,
                } = this.state;
                return (
      <div className="container">
        <div className="row justify-content-center">
          <div className="col-md-8">
            <div className="card">
              <div className="card-header">Register</div>

              <div className="card-body">
                <form
                  method="POST"
                  onSubmit={this.onSubmit}
                  encType="multipart/form-data"
                >
                  <div className="form-group row">
                    <label
                      htmlFor="login"
                      className="col-md-4 col-form-label text-
md-right"
                    >
                      Login
                    </label>

                    <div className="col-md-6">
                      <input
                        id="login"
                        type="text"
                        className="form-control"
                        name="login"
                        required
                        autoComplete="login"
                        autoFocus
                        onChange={this.onBaseInputChange}
                        value={this.state.login}
                      />
                    </div>
                  </div>

                  <div className="form-group row">
                    <label
                      htmlFor="email"
                      className="col-md-4 col-form-label text-
md-right"
                    >
                      E-Mail Address
                    </label>

                    <div className="col-md-6">
                      <input
                        id="email"
                        type="email"
                        className={`form-control
${this.setErrorClass(
                          isEmailError
                        )}`}
                        name="email"
                        required
                        autoComplete="email"
                        onChange={this.onEmailChange}
                        value={this.state.email}
                      />
                      {this.setErrorMsg(isEmailError,
emailErrorMsg)}
                    </div>
                  </div>

                  <div className="form-group row">
                    <label
                      htmlFor="password"
                      className="col-md-4 col-form-label text-
md-right"
                    >
                      Password
                    </label>

                    <div className="col-md-6">
                      <input
                        id="password"
                        type="password"
                        className={`form-control
${this.setErrorClass(
                          isPasswordConfirmError
                        )}`}
                        name="password"
                        required
                        autoComplete="new-password"
                        onChange={this.onBaseInputChange}
                        value={this.state.password}
                      />
                      {this.setErrorMsg(
                        isPasswordConfirmError,
                        passwordConfimMsg
```

```
                    )}
                  </div>
                </div>

                <div className="form-group row">
                  <label
                    htmlFor="passwordConfirm"
                    className="col-md-4 col-form-label text-
md-right"
                  >
                    Confirm Password
                  </label>

                  <div className="col-md-6">
                    <input
                      id="passwordConfirm"
                      type="password"
                      className="form-control"
                      name="passwordConfirm"
                      required
                      autoComplete="new-password"

onChange={this.onPasswordConfirmChange}
                      value={this.state.passwordConfirm}
                    />
                  </div>
                </div>

                <hr className="my-4" />

                <div className="form-group row">
                  <label
                    htmlFor="lastName"
                    className="col-md-4 col-form-label text-
md-right"
                  >
                    Прізвище
                  </label>

                  <div className="col-md-6">
                    <input
                      id="lastName"
                      type="text"
                      className="form-control"
                      name="lastName"
                      required
                      autoComplete="lastName"
                      autoFocus
                      onChange={this.onBaseInputChange}
                      value={this.state.lastName}
                    />
                  </div>
                </div>

                <div className="form-group row">
                  <label
                    htmlFor="firstName"
                    className="col-md-4 col-form-label text-
md-right"
                  >
                    Ім'я
                  </label>

                  <div className="col-md-6">
                    <input
                      id="firstName"
                      type="text"
                      className="form-control"
                      name="firstName"
                      required
                      autoComplete="firstName"
                      autoFocus
                      onChange={this.onBaseInputChange}
                      value={this.state.firstName}
                    />
                  </div>
                </div>

                <div className="form-group row">
                  <label
                    htmlFor="fatherName"
                    className="col-md-4 col-form-label text-
md-right"
                  >
                    По-батькові
                            </label>

                            <div className="col-md-6">
                              <input
                                id="fatherName"
                                type="text"
                                className="form-control"
                                name="fatherName"
                                required
                                autoComplete="fatherName"
                                autoFocus
                                onChange={this.onBaseInputChange}
                                value={this.state.fatherName}
                              />
                            </div>
                          </div>

                          <div className="form-group row">
                            <label
                              htmlFor="about"
                              className="col-md-4 col-form-label text-
md-right"
                            >
                              Про себе
                            </label>

                            <div className="col-md-6">
                              <textarea
                                id="about"
                                type="text"
                                className="form-control"
                                name="about"
                                autoComplete="about"
                                autoFocus
                                onChange={this.onBaseInputChange}
                                value={this.state.about}
                              ></textarea>
                            </div>
                          </div>

                          <div className="form-group row">
                            <label
                              htmlFor="country"
                              className="col-md-4 col-form-label text-
md-right"
                            >
                              Країна
                            </label>
                            <div className="col-md-6">
                              <select
                                id="country"
                                className="form-control"
                                name="country"
                                required
                                autoFocus
                                onChange={this.onCountryChange}
                                value={this.state.country}
                              >
                                {countries.map((country) => (
                                  <option              key={country.id}
value={country.id}>
                                    {country.name}
                                  </option>
                                ))}
                              </select>
                            </div>
                          </div>

                          <div className="form-group row">
                            <label
                              htmlFor="city"
                              className="col-md-4 col-form-label text-
md-right"
                            >
                              Місто
                            </label>
                            <div className="col-md-6">
                              <select
                                id="city"
                                className="form-control"
                                name="city"
                                required
                                autoFocus
                                onChange={this.onCityChange}
                                value={this.state.city}
                              >
                                {cities.map((city) => (
                                  <option                key={city.id}
value={city.id}>
                                    {city.name}
                                  </option>
                                ))}
                              </select>
                            </div>
                          </div>
```

```
                <div className="form-group row">
                  <label
                    htmlFor="street"
                    className="col-md-4 col-form-label text-
md-right"
                  >
                    Вулиця
                  </label>

                  <div className="col-md-6">
                    <input
                      id="street"
                      type="text"
                      className="form-control"
                      name="street"
                      required
                      autoComplete="street"
                      autoFocus
                      onChange={this.onBaseInputChange}
                      value={this.state.street}
                    />
                  </div>
                </div>

                <div className="form-group row">
                  <label
                    htmlFor="house"
                    className="col-md-4 col-form-label text-
md-right"
                  >
                    Будинок
                  </label>

                  <div className="col-md-6">
                    <input
                      id="house"
                      type="text"
                      className="form-control"
                      name="house"
                      required
                      autoComplete="house"
                      autoFocus
                      onChange={this.onBaseInputChange}
                      value={this.state.house}
                    />
                  </div>
                </div>

                <div className="form-group row">
                  <label
                    htmlFor="flat"
                    className="col-md-4 col-form-label text-
md-right"
                  >
                    Номер квартири
                  </label>

                  <div className="col-md-6">
                    <input
                      id="flat"
                      type="text"
                      className="form-control"
                      name="flat"
                      required
                      autoComplete="flat"
                      autoFocus
                      onChange={this.onBaseInputChange}
                      value={this.state.flat}
                    />
                  </div>
                </div>

                <div className="form-group row">
                  <label
                    htmlFor="phoneNumber"
                              className="col-md-4 col-form-
label text-md-right"
                            >
                              Номер телефону
                            </label>

                            <div className="col-md-6">
                              <input
                                id="phoneNumber"
                                type="text"
                                className="form-control"
                                name="phoneNumber"
                                required
                                autoComplete="phoneNumber"
                                autoFocus
                                onChange={this.onBaseInputChange}
                                value={this.state.phoneNumber}
                              />
                            </div>
                          </div>

                          <div className="form-group row">
                            <label
                              htmlFor="clinic"
                              className="col-md-4 col-form-label text-
md-right"
                            >
                              Клініка
                            </label>
                            <div className="col-md-6">
                              <select
                                id="clinic"
                                className="form-control"
                                name="clinic"
                                required
                                autoFocus
                                onChange={this.onClinicChange}
                                value={this.state.clinic}
                              >
                                {clinics.map((clinic) => (
                                  <option
                                    key={clinic.clinic_id}
                                    value={clinic.clinic_id}
                                  >
                                    {clinic.clinic_name}
                                  </option>
                                ))}
                              </select>
                            </div>
                          </div>

                          <div className="form-group row">
                            <label
                              htmlFor="job"
                              className="col-md-4 col-form-label text-
md-right"
                            >
                              Посада
                            </label>
                            <div className="col-md-6">
                              <select
                                id="job"
                                className="form-control"
                                name="job"
                                required
                                autoFocus
                                onChange={this.onBaseInputChange}
                                value={this.state.job}
                              >
                                {jobs.map((job) => (
                                  <option key={job.id} value={job.id}>
                                    {job.name}
                                  </option>
                                ))}
                              </select>
                            </div>
                          </div>

                          <div className="form-group row">
                            <label
                              htmlFor="department"
                              className="col-md-4 col-form-label text-
md-right"
                            >
                              Відділ
                            </label>
                            <div className="col-md-6">
                              <select
                                id="department"
                                className="form-control"
                                name="department"
                                required
                                autoFocus
                                onChange={this.onDepartmentChange}
                                value={this.state.department}
                              >
                                {departments.map((department) => (
                                  <option           key={department.id}
value={department.id}>
                                    {department.name}
                                  </option>
                                ))}
                              </select>
                            </div>
```

```
                </div>

                <div className="form-group row">
                  <label
                    htmlFor="room"
                    className="col-md-4 col-form-label text-
md-right"
                  >
                    Кабінет
                  </label>
                  <div className="col-md-6">
                    <select
                      id="room"
                      className="form-control"
                      name="room"
                      required
                      autoFocus
                      onChange={this.onRoomChange}
                    >
                      {rooms.map((room) => (
                        <option                key={room.id}
value={room.id}>
                          {room.number}
                        </option>
                      ))}
                    </select>
                  </div>
                </div>

                <div className="form-group row">
                  <label
                    htmlFor="image"
                    className="col-md-4 col-form-label text-
md-right"
                  >
                    Фото
                  </label>

                  <div className="col-md-6">
                    <input
                      id="image"
                      ref={this.imageFileRef}
                      type="file"
                      className="form-control-file"
                      name="image"
                      onChange={this.onImageChange}
                    />
                  </div>
                </div>
                <div className="form-group row mb-0">
                  {this.setOverralErrorMsg(isOverallError,
overallErrorMsg)}
                  <div className="col-md-6 offset-md-4">
                    <button   type="submit"   className="btn
btn-primary mr-2">
                                Register
                              </button>
                    <Link  to="/"  className="btn  btn-danger"
role="button">
                      Cancel
                    </Link>
                  </div>
                </div>
                {this.redirect()}
              </form>
            </div>
          </div>
        </div>
      </div>
    );
  }
}

Register.propTypes = {
  isAuthenticated: PropTypes.bool,
  error: PropTypes.object.isRequired,
  register: PropTypes.func.isRequired,
  city: PropTypes.object.isRequired,
  getCities: PropTypes.func.isRequired,
  clinic: PropTypes.object.isRequired,
  getClinics: PropTypes.func.isRequired,
  country: PropTypes.object.isRequired,
  getCountries: PropTypes.func.isRequired,
  job: PropTypes.object.isRequired,
  getJobs: PropTypes.func.isRequired,
  department: PropTypes.object.isRequired,
  getDepartments: PropTypes.func.isRequired,
  auth: PropTypes.object.isRequired,
  getRegisterData: PropTypes.func.isRequired,
};

const mapStateToProps = (state) => ({
  city: state.city,
  country: state.country,
  error: state.error,
  clinic: state.clinic,
  job: state.job,
  department: state.department,
  auth: state.auth,
  room: state.room,
});

export default connect(mapStateToProps, {
  register,
  getCities,
  getCountries,
  getClinics,
  getJobs,
  getDepartments,
  getRegisterData,
  getRooms,
})(Register);
```

Лістинг visitActions.js

```
import {
  GET_VISITS,
  ADD_VISIT,
  DELETE_VISIT,
  VISITS_LOADING,
} from "./visitTypes";
import { GET_ERRORS } from "../error/errorTypes";
import { returnErrors } from "../error/errorActions";
import axios from "axios";

export const getVisits = (user) => (dispatch) => {
  dispatch(setVisitsLoading());
  axios
    .get("/api/visits", {
      params: {
        ...user,
      },
    })
    .then((res) => {
      dispatch({
        type: GET_VISITS,
        payload: res.data,
      });
    })
    .catch((err) => {
      dispatch(
        returnErrors(err.response.data,    err.response.status,
"GET_VISITS ERROR")
      );
    });
};

export const findVisits = (data, user) => (dispatch) => {
  const query = {
    params: {
      ...data,
      ...user,
    },
  };
  axios.get("/api/visits", query).then((res) => {
    dispatch({
      type: GET_VISITS,
      payload: res.data,
    });
  });
};

export const createVisit = (data) => (dispatch) => {
  const config = {
    headers: {
      "Content-Type": "application/json",
    },
  };

  const body = JSON.stringify(data);
  dispatch(setVisitsLoading());
  axios.post("/api/visits", body, config).then((res) => {
    console.log("create visit", res);
    dispatch({
      type: ADD_VISIT,
      payload: res.data,
    });
  });
};
```

```
export const setVisitsLoading = () => {
  return {
    type: VISITS_LOADING,
  };
};
```

Лістинг visitReducer.js

```
import {
  GET_VISITS,
  ADD_VISIT,
  DELETE_VISIT,
  VISITS_LOADING,
} from "../actions/visit/visitTypes";

const initialState = {
  visits: [],
  loading: false,
  createData: [],
};

export default function (state = initialState, action) {
  switch (action.type) {
    case GET_VISITS:
      return {
        ...state,
        visits: action.payload,
        loading: false,
      };
    case ADD_VISIT:
      return {
        ...state,
        loading: false,
      };
    case VISITS_LOADING:
      return {
        ...state,
        loading: true,
      };
    default:
      return state;
  }
}
```

Лістинг visits.js

```
const express = require("express");
const router = express.Router();
const conn = require("../../config/db");

// @route GET /api/visits
router.get("/", (req, res) => {
  const { search, category, id } = req.query;
  let findStr = "";
  if (search) {
    findStr = `AND ${category} LIKE "%${search}%"`;
  }

  conn.query(
    "SELECT p.id AS patientId, r.id AS roomId, " +
      "cl.id AS clinicId, pr.id AS presenceId FROM patients p " +
      "INNER JOIN presences pr ON pr.patient_id = p.id " +
      "INNER JOIN users u ON u.id = pr.doctor_id " +
      "INNER JOIN employees e ON e.id = u.employee_id " +
      "INNER JOIN rooms r ON e.room_id = r.id " +
      "INNER JOIN departments d ON r.department_id = d.id " +
      "INNER JOIN clinics cl ON d.clinic_id = cl.id " +
      `WHERE pr.doctor_id = ${id} ${findStr} ` +
      "ORDER BY pr.updated_at DESC;",
    (err, results, fields) => {
      if (err) throw err;
      let visits = [];
      const getVisits = new Promise(async (resolve, reject) => {
        for (const result of results) {
          const patient =
            "SELECT p.first_name AS firstName, p.last_name AS lastName, " +
            "p.father_name AS fatherName, p.phone_number AS phoneNumber, " +
            "p.street, p.house, p.flat, c.name AS city, p.id " +
            "FROM patients p INNER JOIN cities c ON c.id = p.city_id " +
            `WHERE p.id = ${result.patientId};`;
          const presence =
            "SELECT p.start_at AS startAt, p.end_at AS endAt, p.id " +
            `FROM presences p WHERE p.id = ${result.presenceId};`;
          const clinic =
            "SELECT cl.name, cl.id FROM clinics cl " +
            `WHERE cl.id = ${result.clinicId};`;
          const room =
            `SELECT r.number, r.id FROM rooms r ` +
            `WHERE r.id = ${result.roomId};`;
          const symptoms =
            "SELECT s.id, s.name, ps.description, " +
            "ps.date_plan as datePlan, ps.date_fact AS dateFact " +
            "FROM symptoms s " +
            "INNER JOIN presence_symptom ps ON ps.symptom_id = s.id " +
            "INNER JOIN presences p ON p.id = ps.presence_id " +
            `WHERE p.id = ${result.presenceId};`;
          const diagnosis =
            "SELECT d.id, d.name, pd.description, pd.date_plan AS datePlan, " +
            "pd.date_fact AS dateFact FROM diseases d " +
            "INNER JOIN presence_disease pd ON pd.disease_id = d.id " +
            "INNER JOIN presences p ON p.id = pd.presence_id " +
            `WHERE p.id = ${result.presenceId}; `;
          const medicaments =
            "SELECT m.id, m.name, pm.description, pm.date_plan AS datePlan, " +
            "pm.date_fact as dateFact " +
            "FROM medicaments m " +
            "INNER JOIN presence_medicament pm ON pm.medicament_id = m.id " +
            "INNER JOIN presences p ON p.id = pm.presence_id " +
            `WHERE p.id = ${result.presenceId}; `;
          const procedures =
            "SELECT p.id, p.name, pp.description, " +
            "pp.date_plan AS datePlan, pp.date_fact AS dateFact FROM procedures p " +
            "INNER JOIN presence_procedure pp ON pp.procedure_id = p.id " +
            "INNER JOIN presences pa ON pa.id = pp.presence_id " +
            `WHERE pa.id = ${result.presenceId}; `;
          const treatments =
            "SELECT t.id, t.name, pt.description, " +
            "pt.date_plan AS datePlan, pt.date_fact AS dateFact FROM treatments t " +
            "INNER JOIN presence_treatment pt ON pt.treatment_id = t.id " +
            "INNER JOIN presences p ON p.id = pt.presence_id " +
            `WHERE p.id = ${result.presenceId}; `;
          const results = await conn
            .promise()
            .query(
              `${patient} ${presence} ${clinic} ${room} ${symptoms} ${diagnosis} ${medicaments} ${procedures} ${treatments}`
            );
          visits.push({
            patient: results[0][0][0],
            presence: results[0][1][0],
            clinic: results[0][2][0],
            room: results[0][3][0],
            symptoms: results[0][4],
            diagnosis: results[0][5],
            medicaments: results[0][6],
            procedures: results[0][7],
            treatments: results[0][8],
          });
        }
        resolve(visits);
      }).then((visits) => {
        return res.json(visits);
      });
    }
```

```
  );
});

// @route POST /api/visits
router.post("/", (req, res) => {
  const data = req.body;
  if (!{ ...data }) {
    return res.status(400).json({ msg: "Please enter all
fields" });
  }
  const newArriveAt = data.arrivedAt.replace("T", " ");
  const newDepartureAt = data.departureAt.replace("T", " ");
  const presenceQuery =
    "INSERT INTO presences (patient_id, " +
    "doctor_id, start_at, end_at) VALUES (" +
    `${data.patientId}, ${data.userId}, ` +
    `"${newArriveAt}", "${newDepartureAt}"); `;
  conn.query(presenceQuery, (err, presenceResults, fields) =>
{
    if (err) return res.status(400).json(err);

    const writeToPivotTables = new Promise(async (resolve,
reject) => {
      try {
        for (const symptom of data.symptoms) {
          let symptomToInsert;
          const results = await conn
            .promise()
            .query(
              "SELECT id, name FROM symptoms WHERE " +
                `name = "${symptom.name}"; `
            );
          if (results[0].length > 0) {
            symptomToInsert = { ...symptom, id:
results[0][0].id };
          } else {
            const newSymptom = await conn
              .promise()
              .query(`INSERT INTO symptoms (name) VALUES
("${symptom.name}")`);
            symptomToInsert = { ...symptom, id:
newSymptom[0].insertId };
          }
          const presenceSymptom = await conn
            .promise()
            .query(
              "INSERT INTO presence_symptom (presence_id, " +
                "symptom_id, description, date_plan,
date_fact) " +
                `VALUES ($${presenceResults.insertId},
${symptomToInsert.id}, ` +
                `"${symptomToInsert.description}",
"${newArriveAt}", "${newDepartureAt}"); `
            );
        }
        for (const diagnos of data.diagnosis) {
          let diagnosToInsert;
          const results = await conn
            .promise()
            .query(
              "SELECT id, name FROM diseases WHERE " +
                `name = "${diagnos.name}"; `
            );
          if (results[0].length > 0) {
            diagnosToInsert = { ...diagnos, id:
results[0][0].id };
          } else {
            const newDiagnos = await conn
              .promise()
              .query(`INSERT INTO diseases (name) VALUES
("${diagnos.name}")`);
            diagnosToInsert = { ...diagnos, id:
newDiagnos[0].insertId };
          }
          const presenceDisease = await conn
            .promise()
                        .query(
                          "INSERT INTO presence_disease
(presence_id, " +
                            "disease_id, description,
date_plan, date_fact) " +
                `VALUES ($${presenceResults.insertId},
${diagnosToInsert.id}, ` +
                `"${diagnosToInsert.description}",
"${newArriveAt}", "${newDepartureAt}"); `
            );
        }
        for (const medicament of data.medicaments) {
          let medicamentToInsert;
          const results = await conn
            .promise()
            .query(
              "SELECT id, name FROM medicaments WHERE " +
                `name = "${medicament.name}"`
            );
          if (results[0].length > 0) {
            medicamentToInsert = { ...medicament, id:
results[0][0].id };
          } else {
            const newMedicament = await conn
              .promise()
              .query(
                `INSERT INTO medicaments(name) VALUES
("${medicament.name}")`
              );
            medicamentToInsert = {
              ...medicament,
              id: newMedicament[0].insertId,
            };
          }
          const presenceMedicament = await conn
            .promise()
            .query(
              "INSERT INTO presence_medicament (presence_id, "
+
                "medicament_id, description, date_plan,
date_fact) " +
                `VALUES ($${presenceResults.insertId},
${medicamentToInsert.id}, ` +
                `"${medicamentToInsert.description}",
"${newArriveAt}", "${newDepartureAt}"); `
            );
        }
        for (const procedure of data.procedures) {
          let procedureToInsert;
          const results = await conn
            .promise()
            .query(
              "SELECT id, name FROM procedures WHERE " +
                `name = "${procedure.name}"`
            );
          if (results[0].length > 0) {
            procedureToInsert = { ...procedure, id:
results[0][0].id };
          } else {
            const newProcedure = await conn
              .promise()
              .query(
                `INSERT INTO procedures(name) VALUES
("${procedure.name}")`
              );
            procedureToInsert = { ...procedure, id:
newProcedure[0].insertId };
          }
          const presenceProcedure = await conn
            .promise()
            .query(
              "INSERT INTO presence_procedure (presence_id, "
+
                "procedure_id, description, date_plan,
date_fact) " +
                `VALUES ($${presenceResults.insertId},
${procedureToInsert.id}, ` +
                `"${procedureToInsert.description}",
"${newArriveAt}", "${newDepartureAt}"); `
            );
        }
        for (const treatment of data.treatments) {
          let treatmentToInsert;
          const results = await conn
            .promise()
            .query(
              "SELECT id, name FROM treatments WHERE " +
                `name = "${treatment.name}"`
            );
          if (results[0].length > 0) {
            treatmentToInsert = { ...treatment, id:
results[0][0].id };
          } else {
            const newTreatment = await conn
              .promise()
              .query(
                `INSERT INTO treatments(name) VALUES
("${treatment.name}")`
              );
            treatmentToInsert = { ...treatment, id:
newTreatment[0].insertId };
          }
          const presenceTreatment = await conn
            .promise()
```

```
            .query(
              "INSERT INTO presence_treatment (presence_id, " +
                "treatment_id,     description,     date_plan,
date_fact) " +
                `VALUES          (${presenceResults.insertId},
${treatmentToInsert.id}, ` +
                `"${treatmentToInsert.description}",
"${newArriveAt}", "${newDepartureAt}"); `
            );
        }
      } catch (err) {
        reject(err);
      }
      resolve(presenceResults);
    })
      .then((visits) => {
        return res.json(visits);
      })
      .catch((err) => {
        return res.status(400).json(err);
      });
  });
});

module.exports = router;

import React from "react";
import { Link } from "react-router-dom";
import PropTypes from "prop-types";
import { connect } from "react-redux";
import      {      getVisits,      findVisits      }      from
"../../../actions/visit/visitActions";
import          {          addLink          }          from
"../../../actions/navigation/navigationActions";
import moment from "moment";
import TableView from "../Helpers/TableView";
import Loading from "../../modals/Loading";

class Visits extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      loading: true,
      search: "",
      category: "",
    };
  }

  componentDidMount() {
    const { user } = this.props.auth;
    this.props.getVisits(user);
    this.props.addLink({ path: window.location.pathname, name:
"Візити" });
  }

  componentDidUpdate(prevProps) {
    if (prevProps.visit.loading !== this.props.visit.loading)
{
      this.setState({
        loading: this.props.visit.loading,
        category: "p.last_name",
      });
    }
  }

  search = (e) => {
    e.preventDefault();
    const { search, category } = this.state;
    const { user } = this.props.auth;
    this.props.findVisits({ search, category }, user);
  };

  onBaseInputChange = (e) => {
    this.setState({
      ...this.state,
      [e.target.name]: e.target.value,
    });
  };

  render() {
    if (this.state.loading) {
      return <Loading />;
    } else {
      const { user } = this.props.auth;
      const { visits } = this.props.visit;

      return (
        <div className="container">
          <h3 className="text-center mt-3">Візити</h3>
          <div className="row w-100">
                <div className="col-4"></div>
                <div className="col-6 my-3">
            <form   onSubmit={this.search}   className="form-
inline">
              <div className="input-group">
                <select
                  name="category"
                  onChange={this.onBaseInputChange}
                  className="form-control w-25"
                >
                  <option
value="p.last_name">Прізвище</option>
                  <option value="p.first_name">Ім'я</option>
                  <option           value="p.father_name">По-
батькові</option>
                  <option value="p.street">Вулиця</option>
                  <option                 value="pr.id">Номер
картки</option>
                </select>
                <input
                  id="search"
                  name="search"
                  className="form-control  w-50  input-group-
append"
                  value={this.state.search}
                  onChange={this.onBaseInputChange}
                  type="text"
                  placeholder="Search"
                  aria-label="Search"
                />
                <div className="input-group-append">
                  <button     className="btn     btn-success"
type="submit">
                    Find<span            className="glyphicon
glyphicon-search"></span>
                  </button>
                </div>
              </div>
            </form>
          </div>
          <div className="col-4"></div>
        </div>
        {visits.map((visit) => (
          <div    key={visit.presence.id}    className="card
rounded-0 mt-3">
            <div className="row">
              <div className="col pr-0">
                <div  className="card  border-top-0  border-
left-0 rounded-0 container">
                  <div                      className="card-
body">№{visit.presence.id}</div>
                </div>
              </div>
              <div className="col px-0">
                <div  className="card  border-top-0  border-
left-0 border-right-0 rounded-0 container">
                  <div className="card-body">
                    Початок:{" "}

{moment(visit.presence.startAt).format("DD.MM.YYYY H:mm")}
                  </div>
                </div>
              </div>
              <div className="col pl-0">
                <div  className="card  border-top-0  border-
right-0 rounded-0 container">
                  <div className="card-body">
                    Кінець:{" "}
                    {visit.departure_at !== null &&

moment(visit.presence.endAt).format("DD.MM.YYYY H:mm")}
                  </div>
                </div>
              </div>
            </div>
            <div className="row">
              <div className="col pr-0">
                <div  className="card  rounded-0  container
border-left-0 border-bottom-0 border-top-0 border-right-1">
                  <div className="card-body">
                    <h4>Пацієнт</h4>
                    <p>
                      <span className="h5">ПІБ: </span>{" "}
                      {visit.patient.lastName}
{visit.patient.firstName}{" "}
                      {visit.patient.fatherName}
                    </p>
                    <p>
```

```
                        <span  className="h5">Адреса:  </span>
м.
                        {visit.patient.city}, вул.
                        {visit.patient.street}
{visit.patient.house}, кв.{" "}
                        {visit.patient.flat}
                      </p>
                      <p>
                        <span          className="h5">Телефон:
</span>{" "}
                        {visit.patient.phoneNumber}
                      </p>
                    </div>
                  </div>
                </div>
                <div className="col pl-0">
                  <div  className="card  rounded-0  container
border-0">
                    <div className="card-body">
                      <h4>Прийом</h4>
                      <p>
                        <span className="h5">Заклад: </span>
                        {visit.clinic.name}
                      </p>
                      <p>
                        <span className="h5">Кабінет: </span>
                        {visit.room.number}
                      </p>
                      <div className="pt-0">
                        <Link

to={`/patients/${visit.patient.id}/show`}
                          className="btn btn-primary"
                          role="button"
                        >
                          Картка пацієнта
                        </Link>
                      </div>
                    </div>
                  </div>
                </div>
              </div>
              <div className="row">
                <div className="col">
                  <div className="card rounded-0 border-left-0
border-bottom-0 border-top-1 border-right-0 container">
                    <div className="card-body">
                      <TableView          data={visit.symptoms}
name="Симптоми" />
                      <TableView         data={visit.diagnosis}
name="Діагнози" />
                      <TableView       data={visit.medicaments}
name="Медикаменти" />
                      <TableView        data={visit.procedures}
name="Процедури" />
                      <TableView        data={visit.treatments}
name="Лікування" />
                    </div>
                  </div>
                </div>
              </div>
            </div>
          ))}
        </div>
      );
    }
  }
}

Visits.propTypes = {
  auth: PropTypes.object.isRequired,
  visit: PropTypes.object.isRequired,
  getVisits: PropTypes.func.isRequired,
  findVisits: PropTypes.func.isRequired,
  addLink: PropTypes.func.isRequired,
};

const mapStateToProps = (state) => ({
  auth: state.auth,
  visit: state.visit,
});

export    default    connect(mapStateToProps,    {    getVisits,
findVisits, addLink })(
  Visits
);
```

Лістинг VisitCreateForm.js

```
import React from "react";
import PropTypes from "prop-types";
import { connect } from "react-redux";
import moment from "moment";
import { Link, Redirect } from "react-router-dom";
import { withRouter } from "react-router";
import          {          createVisit          }          from
"../../../actions/visit/visitActions";
import CloseButton from "../../modals/CloseButton";

class VisitsCreateForm extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      visit: {
        arrivedAt: moment(),
        departureAt: moment(),
        symptoms: [{ id: 1, name: "", description: "" }],
        diagnosis: [{ id: 1, name: "", description: "" }],
        medicaments: [{ id: 1, name: "", description: "" }],
        procedures: [{ id: 1, name: "", description: "" }],
        treatments: [{ id: 1, name: "", description: "" }],
      },
      isDateErr: false,
      isOverallError: false,
      isComplete: false,
      dateErrMsg: "Дата початку пізніше, ніж дата закінчення",
      overallErrorMsg: "Помилка створення візиту",
    };
  }

  onSubmit = (e) => {
    const  {  isOverallError,  overallErrorMsg,  visit  }  =
this.state;
    e.preventDefault();
    if (isOverallError) {
      alert(overallErrorMsg);
    } else {
      let patientId;
      try {
        patientId = parseInt(this.props.match.params.id);
      } catch (err) {
        console.log(err);
      }
      const userId = this.props.auth.user.id;
      let   symptoms,   diagnosis,   medicaments,   procedures,
treatments;
      if (visit.symptoms[0].name === "") {
        symptoms = [];
      } else {
        symptoms = visit.symptoms;
      }
      if (visit.diagnosis[0].name === "") {
        diagnosis = [];
      } else {
        diagnosis = visit.diagnosis;
      }
      if (visit.medicaments[0].name === "") {
        medicaments = [];
      } else {
        medicaments = visit.medicaments;
      }
      if (visit.procedures[0].name === "") {
        procedures = [];
      } else {
        procedures = visit.procedures;
      }
      if (visit.treatments[0].name === "") {
        treatments = [];
      } else {
        treatments = visit.treatments;
      }
      const nVisit = {
        ...this.state.visit,
        symptoms,
        diagnosis,
        medicaments,
        procedures,
        treatments,
      };
      this.props.createVisit({
        ...nVisit,
        patientId,
        userId,
      });
      this.setState({
        ...this.state,
        isComplete: true,
      });
```

```
    }
  };

  onDepartureAtChange = (e) => {
    const departureAt = e.target.value;
    const { arrivedAt } = this.state;
    if (departureAt <= arrivedAt) {
      this.setState({
        ...this.state,
        isDateErr: true,
        isOverallError: true,
      });
    } else {
      this.setState({
        ...this.state,
        isDateErr: false,
        isOverallError: false,
        visit: {
          ...this.state.visit,
          departureAt,
        },
      });
    }
  };

  setErrorClass = (error) => {
    if (error) {
      return "is-invalid";
    } else {
      return "";
    }
  };

  setErrorMsg = (error, msg) => {
    if (error) {
      return (
        <span className="invalid-feedback" role="alert">
          <strong>{msg}</strong>
        </span>
      );
    } else {
      return <span></span>;
    }
  };

  redirect = () => {
    if (this.state.isComplete) {
      return <Redirect to="/visits" />;
    }
  };

  setOverralErrorMsg = (error, msg) => {
    if (error) {
      return (
        <span     className="text-danger     mx-auto     mb-4"
role="alert">
          <strong>{msg}</strong>
        </span>
      );
    } else {
      return <span></span>;
    }
  };

  onBaseInputChange = (e) => {
    this.setState({
      ...this.state,
      visit: {
        ...this.state.visit,
        [e.target.name]: e.target.value,
      },
    });
  };

  onVisitInputChange = (obj) => {
    const property = Object.keys(obj)[0];
    const field = Object.keys(obj)[1];
    let data = Object.values(obj)[0];
    const  index  =  data.findIndex((item)  =>  item.id  ===
obj.id);
    data[index][field] = obj[field];
    this.setState({
      ...this.state,
      visit: {
        ...this.state.visit,
        [property]: data,
      },
    });
  };

  moreVisitInput = (obj) => {
    const property = Object.keys(obj)[0];
    const data = Object.values(obj)[0];
    this.setState({
      ...this.state,
      visit: {
        ...this.state.visit,
        [property]: [
          ...data,
          {
            id: data[data.length - 1].id + 1,
            name: "",
            description: "",
          },
        ],
      },
    });
  };

  deleteVisitInput = (obj) => {
    const property = Object.keys(obj)[0];
    let data = Object.values(obj)[0];
    if (data.length > 1) {
      data = data.filter((item) => item.id !== obj.id);
      this.setState({
        ...this.state,
        visit: {
          ...this.state.visit,
          [property]: data,
        },
      });
    }
  };

  render() {
    const {
      isDateErr,
      dateErrMsg,
      isOverallError,
      overallErrorMsg,
    } = this.state;
    const {
      arrivedAt,
      departureAt,
      symptoms,
      diagnosis,
      treatments,
      medicaments,
      procedures,
    } = this.state.visit;
    const patientId = this.props.match.params.id;
    return (
      <div className="container">
        <div className="row justify-content-center">
          <div className="col-md-8">
            <div className="card">
              <div  className="card-header">Зареєструвати новий
візит</div>
              <div className="card-body">
                <form onSubmit={this.onSubmit}>
                  <div className="form-group row">
                    <label
                      htmlFor="arrivedAt"
                      className="col-md-4 col-form-label text-
md-right"
                    >
                      Дата початку
                    </label>
                    <div className="col-md-6">
                      <input
                        id="arrivedAt"
                        name="arrivedAt"
                        type="datetime-local"
                        value={arrivedAt}
                        onChange={this.onBaseInputChange}
                        className={`form-control
${this.setErrorClass(
                          isDateErr
                        )}`}
                      />
                      {this.setErrorMsg(isDateErr,
dateErrMsg)}
                    </div>
                  </div>

                  <div className="form-group row">
                    <label
                      htmlFor="departureAt"
                      className="col-md-4 col-form-label text-
md-right"
```

```
                    >
                      Дата закінчення
                    </label>
                    <div className="col-md-6">
                      <input
                        id="departureAt"
                        name="departureAt"
                        type="datetime-local"
                        value={departureAt}
                        onChange={this.onDepartureAtChange}
                        className="form-control"
                      />
                    </div>
                  </div>

                  <div className="form-group-row text-center">
                    <label
                      htmlFor="symptom"
                      className="col-md-8  col-form-label  big-
label text-lg-center"
                    >
                      Симптоми
                    </label>
                  </div>
                  {symptoms.map((symptom) => (
                    <React.Fragment key={symptom.id}>
                      <div className="form-group row">
                        <label
                          htmlFor="symptom"
                          className="col-md-4    col-form-label
text-md-right"
                        >
                          Симптом:
                        </label>
                        <div className="col-md-6">
                          <input
                            id="symptom"
                            type="text"
                            className="form-control"
                            name="symptom"
                            value={symptom.name}
                            onChange={(e) =>
                              this.onVisitInputChange({
                                symptoms,
                                name: e.target.value,
                                id: symptom.id,
                              })
                            }
                          />
                        </div>
                        <CloseButton
                          func={this.deleteVisitInput}
                          args={[{  symptoms,  id:  symptom.id
}]}
                        />
                      </div>
                      <div className="form-group row">
                        <label
                          htmlFor="symptomDescription"
                          className="col-md-4    col-form-label
text-md-right"
                        >
                          Доповнення:
                        </label>
                        <div className="col-md-6">
                          <textarea
                            className="form-control"
                            name="symptomDescription"
                            value={symptom.description}
                            id="symptomDescription"
                            rows="5"
                            onChange={(e) =>

this.onVisitInputChange({
                                        symptoms,
                                        description:

e.target.value,
                                        id: symptom.id,
                                      })
                                    }
                                  ></textarea>
                                </div>
                              </div>
                            </React.Fragment>
                          ))}

                          <div className="form-group row">
                  <label
                    htmlFor="moreSymptom"
                    className="col-md-4 col-form-label text-
md-right"
                  ></label>
                  <div className="col-md-6">
                    <button
                      className="btn btn-primary"
                      onClick={(e)  =>  this.moreVisitInput({
symptoms })}
                      id="moreSymptom"
                      type="button"
                    >
                      Ще симптом
                    </button>
                  </div>
                </div>

                <div className="form-group-row text-center">
                  <label
                    htmlFor="diagnos"
                    className="col-md-8  col-form-label  big-
label text-md-center"
                  >
                    Діагнози
                  </label>
                </div>
                {diagnosis.map((diagnos) => (
                  <React.Fragment key={diagnos.id}>
                    <div className="form-group row">
                      <label
                        htmlFor="diagnos"
                        className="col-md-4    col-form-label
text-md-right"
                      >
                        Діагноз:
                      </label>
                      <div className="col-md-6">
                        <input
                          id="diagnos"
                          type="text"
                          className="form-control"
                          name="diagnos"
                          value={diagnos.name}
                          onChange={(e) =>
                            this.onVisitInputChange({
                              diagnosis,
                              name: e.target.value,
                              id: diagnos.id,
                            })
                          }
                        />
                      </div>
                      <CloseButton
                        func={this.deleteVisitInput}
                        args={[{  diagnosis,  id:  diagnos.id
}]}
                      />
                    </div>
                    <div className="form-group row">
                      <label
                        htmlFor="diagnosDescription"
                        className="col-md-4    col-form-label
text-md-right"
                      >
                        Доповнення:
                      </label>
                      <div className="col-md-6">
                        <textarea
                          className="form-control"
                          name="diagnosDescription"
                          value={diagnos.description}
                          id="diagnosDescription"
                          rows="5"
                          onChange={(e) =>
                            this.onVisitInputChange({
                              diagnosis,
                              description: e.target.value,
                              id: diagnos.id,
                            })
                          }
                        ></textarea>
                      </div>
                    </div>
                  </React.Fragment>
                ))}

                <div className="form-group row">
                  <label
                    htmlFor="moreDiagnos"
                    className="col-md-4 col-form-label text-
md-right"
```

```
                    ></label>
                    <div className="col-md-6">
                      <button
                        className="btn btn-primary"
                        onClick={(e) => this.moreVisitInput({
diagnosis })}
                        id="moreDiagnos"
                        type="button"
                      >
                        Ще діагноз
                      </button>
                    </div>
                  </div>

                  <div className="form-group-row text-center">
                    <label
                      htmlFor="medicament"
                      className="col-md-8 col-form-label big-
label text-md-center"
                    >
                      Медикаменти
                    </label>
                  </div>
                  {medicaments.map((medicament) => (
                    <React.Fragment key={medicament.id}>
                      <div className="form-group row">
                        <label
                          htmlFor="medicament"
                          className="col-md-4    col-form-label
text-md-right"
                        >
                          Медикамент:
                        </label>
                        <div className="col-md-6">
                          <input
                            id="medicament"
                            type="text"
                            className="form-control"
                            name="medicament"
                            value={medicament.name}
                            onChange={(e) =>
                              this.onVisitInputChange({
                                medicaments,
                                name: e.target.value,
                                id: medicament.id,
                              })
                            }
                          />
                        </div>
                        <CloseButton
                          func={this.deleteVisitInput}
                                    args={[{ medicaments, id:
medicament.id }]}
                                  />
                                </div>
                                <div    className="form-group
row">
                                  <label

htmlFor="medicamentDescription"
                                    className="col-md-4   col-
form-label text-md-right"
                                  >
                                    Доповнення:
                        </label>
                        <div className="col-md-6">
                          <textarea
                            className="form-control"
                            name="medicamentDescription"
                            value={medicament.description}
                            id="medicamentDescription"
                            rows="5"
                            onChange={(e) =>
                              this.onVisitInputChange({
                                medicaments,
                                description: e.target.value,
                                id: medicament.id,
                              })
                            }
                          ></textarea>
                        </div>
                      </div>
                    </React.Fragment>
                  ))}

                  <div className="form-group row">
                    <label
                      htmlFor="moreMedicament"
                      className="col-md-4 col-form-label text-
md-right"
                    ></label>
                    <div className="col-md-6">
                      <button
                        className="btn btn-primary"
                        onClick={(e) => this.moreVisitInput({
medicaments })}
                        id="moreMedicament"
                        type="button"
                      >
                        Ще медикамент
                      </button>
                    </div>
                  </div>

                  <div className="form-group-row text-center">
                    <label
                      htmlFor="procedure"
                      className="col-md-8 col-form-label big-
label text-md-center"
                    >
                      Процедури
                    </label>
                  </div>
                  {this.state.visit.procedures.map((procedure)
=> (
                    <React.Fragment key={procedure.id}>
                      <div className="form-group row">
                        <label
                          htmlFor="procedure"
                          className="col-md-4    col-form-label
text-md-right"
                        >
                          Процедура:
                        </label>
                        <div className="col-md-6">
                          <input
                            id="procedure"
                            type="text"
                            className="form-control"
                            name="procedure"
                            value={procedure.name}
                            onChange={(e) =>
                              this.onVisitInputChange({
                                procedures,
                                name: e.target.value,
                                id: procedure.id,
                              })
                            }
                          />
                        </div>
                        <CloseButton
                          func={this.deleteVisitInput}
                          args={[{       procedures,       id:
procedure.id }]}
                        />
                      </div>
                      <div className="form-group row">
                        <label
                          htmlFor="procedureDescription"
                          className="col-md-4    col-form-label
text-md-right"
                        >
                          Доповнення:
                        </label>
                        <div className="col-md-6">
                          <textarea
                            className="form-control"
                            name="procedureDescription"
                            value={procedure.description}
                            id="procedureDescription"
                            rows="5"
                            onChange={(e) =>
                              this.onVisitInputChange({
                                procedures,
                                description: e.target.value,
                                id: procedure.id,
                              })
                            }
                          ></textarea>
                        </div>
                      </div>
                    </React.Fragment>
                  ))}

                  <div className="form-group row">
                    <label
                      htmlFor="moreProcedure"
                      className="col-md-4 col-form-label text-
md-right"
                    ></label>
```

```
                    <div className="col-md-6">
                      <button
                        className="btn btn-primary"
                        onClick={(e)  =>  this.moreVisitInput({
procedures })}
                        id="moreProcedure"
                        type="button"
                      >
                        Ще процедура
                      </button>
                    </div>
                  </div>

                  <div className="form-group-row text-center">
                    <label
                      htmlFor="treatment"
                      className="col-md-8  col-form-label  big-
label text-md-center"
                    >
                      Схеми лікувань
                    </label>
                  </div>
                  {this.state.visit.treatments.map((treatment)
=> (
                    <React.Fragment key={treatment.id}>
                      <div className="form-group row">
                        <label
                          htmlFor="treatment"
                          className="col-md-4    col-form-label
text-md-right"
                        >
                          Схема лікування:
                        </label>
                        <div className="col-md-6">
                          <input
                            id="treatment"
                            type="text"
                            className="form-control"
                            name="treatment"
                            value={treatment.name}
                                       onChange={(e) =>
                              this.onVisitInputChange({
                                treatments,
                                name: e.target.value,
                                id: treatment.id,
                              })
                            }
                          />
                        </div>
                        <CloseButton
                          func={this.deleteVisitInput}
                          args={[{       treatments,       id:
treatment.id }]}
                        />
                      </div>
                      <div className="form-group row">
                        <label
                          htmlFor="treatmentDescription"
                          className="col-md-4    col-form-label
text-md-right"
                        >
                          Доповнення:
                        </label>
                        <div className="col-md-6">
                          <textarea
                            className="form-control"
                            name="treatmentDescription"
                            value={treatment.description}
                            id="treatmentDescription"
                            rows="5"
                            onChange={(e) =>
                              this.onVisitInputChange({
                                treatments,
                                description: e.target.value,
                                id: treatment.id,
                              })
                            }
                          ></textarea>
                        </div>
                      </div>
                    </React.Fragment>
                  ))}

                  <div className="form-group row">
                    <label
                      htmlFor="moreTreatment"
                      className="col-md-4 col-form-label text-
md-right"
                    ></label>
                    <div className="col-md-6">
                      <button
                        className="btn btn-primary"
                        onClick={(e)  =>  this.moreVisitInput({
treatments })}
                        id="moreTreatment"
                        type="button"
                      >
                        Ще схема лікування
                      </button>
                    </div>
                  </div>

                  <div className="form-group row mb-0">
                    <div  className="col-md-6  offset-md-4  mt-
5">
                      {this.setOverralErrorMsg(isOverallError,
overallErrorMsg)}
                      <button   type="submit"   className="btn
btn-primary mr-2">
                        Зареєструвати
                      </button>
                      <Link
                        to={`/patients/${patientId}/show`}
                        className="btn btn-danger"
                        role="button"
                      >
                        Відмінити
                      </Link>
                    </div>
                  </div>
                  {this.redirect()}
                </form>
              </div>
            </div>
                    </div>
                  </div>
      </div>
    );
  }
}

VisitsCreateForm.propTypes = {
  auth: PropTypes.object.isRequired,
  createVisit: PropTypes.func.isRequired,
};

const mapStateToProps = (state) => ({
  auth: state.auth,
});

export default withRouter(
  connect(mapStateToProps, { createVisit })(VisitsCreateForm)
);
```

Додаток Д

Презентаційний матеріал

Кваліфікаційна робота бакалавра

Автоматизована система допомоги в діагностуванні хвороб терапевтом

Виконав: студент групи КН-17-2 *І.О. Вишинський*.
Керівник: к.т.н., доцент кафедри КНІТ *О.В. Мазурець*

Актуальність

Автоматизована система допомоги в діагностуванні може суттєво полегшити роботу лікарів, а також зменшити кількість лікарських помилок. З її допомогою лікарі зможуть зменшити об'єм паперової роботи, а також отримувати інформацію про вірогідний діагноз, який визначить аналітична система. Це дозволить лікарям швидше та точніше ставити діагнози.

Найбільш зручно використовувати таку систему буде в поліклініках. Терапевт, який там працює має постійний потік пацієнтів та багато паперової роботи. Тому в поліклініках дана система буде дуже корисною.

Завдання

Метою кваліфікаційної роботи бакалавра є розробка автоматизованої системи допомоги в діагностуванні хвороби терапевтом на платформі JavaScript, що виконує наступні основні функції:

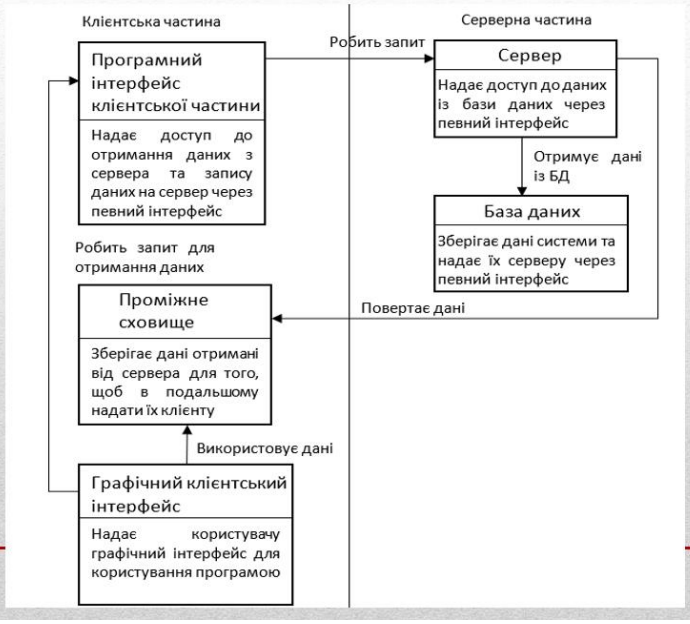
- робота з відкритою інформацією (виведення інформації про лікарів та поліклініки, зареєстровані в системі);
- допомога в діагностуванні хвороби;
- робота з візитами (додавання симптомів пацієнта, діагнозів, направлень на процедури, рецептів на медикаменти);
- робота з картками пацієнтів (додавання симптомів пацієнта, діагнозів, направлень на процедури, рецептів на медикаменти, редагування його особистої інформації);
- робота з особистими даними (реєстрація в системі, авторизація, редагування даних профілю);
- робота з внутрішніми даними системи (хвороби, симптоми, медикаменти, процедури, працівники, пацієнти, палати);
- робота зі статистикою роботи терапевта (по симптомах пацієнта, діагнозах, направленнях на процедури, рецептах на медикаменти).

Функціональна структура

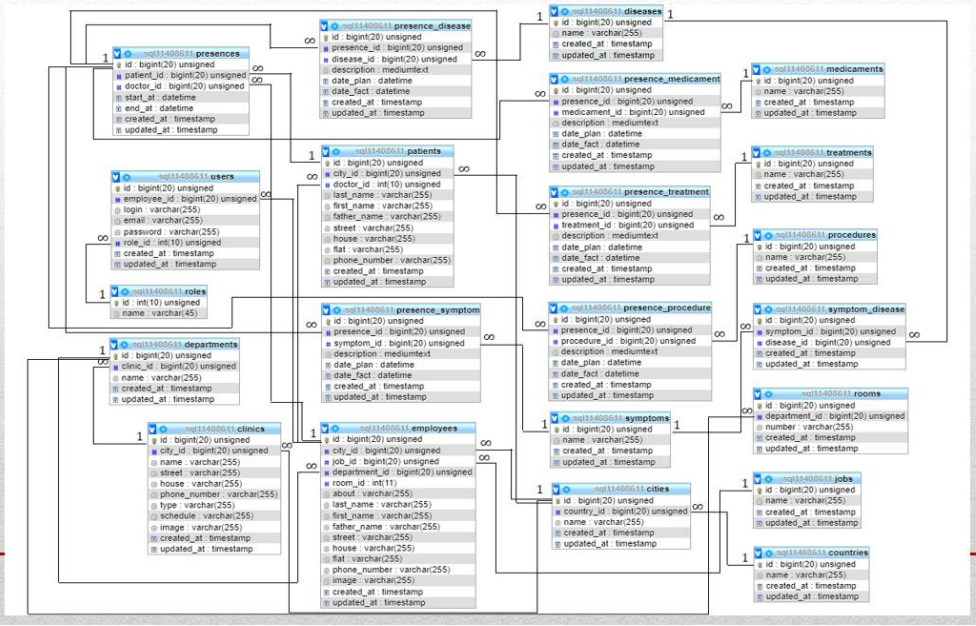
При розробці автоматизованої системи допомоги в діагностуванні хвороби терапевтом були автоматизовані наступні бізнес-процеси:

- бізнес-процес «Робота з відкритою інформацією»;
- бізнес-процес «Робота з візитами»;
- бізнес-процес «Робота з картками пацієнтів»;
- бізнес-процес «Робота з особистими даними»;
- бізнес-процес «Робота з внутрішніми даними системи»;
- бізнес-процес «Робота зі статистикою роботи терапевта»;
- бізнес-процес «Робота з системою допомоги в діагностуванні».

Архітектура проекту



Структура бази даних



Платформа

JavaScript (JS) – скриптова мова програмування з динамічною типізацією. В цих аспектах вона схожа на PHP, але відрізняється від C#. JavaScript поєднує в собі декілька парадигм, таких як: об’єктно-орієнтоване програмування та функціональне програмування.

Найчастіше, дана мова програмування використовується для клієнтської частини додатку, для взаємодії з користувачем, динамічної зміни контенту веб-сторінки, створення інтерактивності.

Але крім цього, JavaScript також можна використовувати і для створення серверної частини. За допомогою фреймворків, доступних даній платформі, це робити доволі зручно.

Мова програмування

Найчастіше, дана мова програмування використовується для клієнтської частини додатку, для взаємодії з користувачем, динамічної зміни контенту веб-сторінки, створення інтерактивності, взаємодії з DOM. Але крім цього, JavaScript також можна використовувати і для створення серверної частини.

Працюючи з JavaScript на серверній частині додатку, ми, в основному, передаєм дані у форматі JSON на сторону клієнта. Завдяки близькості формату JSON цієї мови програмування, робити це відносно просто. Також потрібно розуміти, що за допомогою JavaScript можна писати не тільки серверну та клієнтську частини веб-додатку, а також мобільні додатки та додатки для персональних комп’ютерів.

Фреймворк

Для виконання даної КРБ було обрано 3 фреймворки. Один для клієнтської частини – React, один для серверної частини – Express, один для зв'язування клієнтської та серверної частини – Redux.

React – JavaScript-бібліотека з відкритим вихідним кодом. Вона призначена для розробки клієнтської частини додатку.

Для більш зручного управління даними додатку використовується фреймворк Redux. Він дозволяє поділити роботу з даними на певні функціональні блоки, кожен з яких буде складатися з контейнера та контролера. Контейнер буде зберігати необхідні дані. А контролер посилати запити на сервер для того, щоб отримати певні дані від моделі. Даний фреймворк працює у клієнтській частині додатку.

Для роботи серверної частини було обрано фреймворк Express. Це фреймворк, який працює в середині середовища Node.js, тобто з його допомогою можна створювати серверну частину додатку. Даний фреймворк не заставляє програміста використовувати якийсь певний архітектурний шаблон, типу MVC. І це є однією з переваг даного фреймворку.

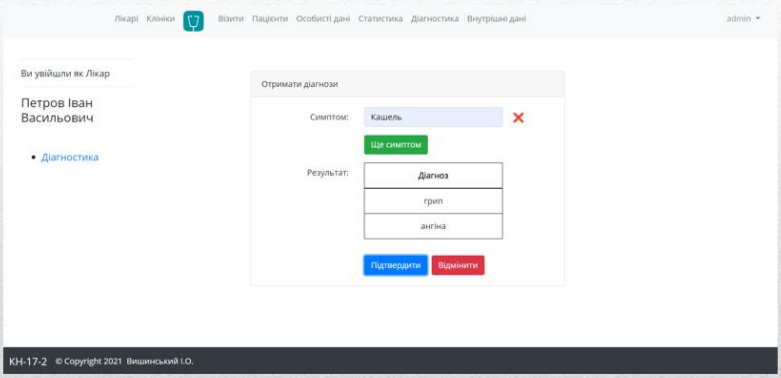
СКБД

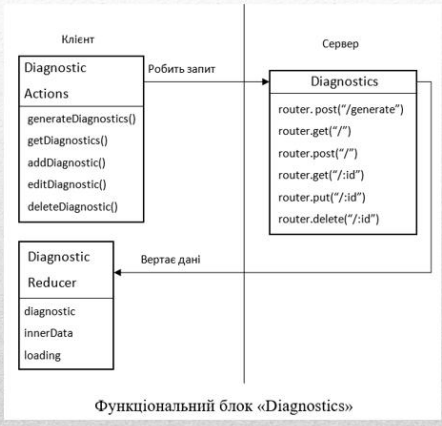
Для виконання даної роботи було обрано СКБД MySQL.

MySQL – система керування реляційними базами даних.

MySQL працює з багатьма фреймворками, працює на багатьох платформах, має якісну документацію, високі стандарти безпеки, велику кількість розробників, які використовують дану СКБД. Крім цього завдяки популярності даної СКБД, є висока ймовірність, що вона буде і надалі дороблятися її розробниками.

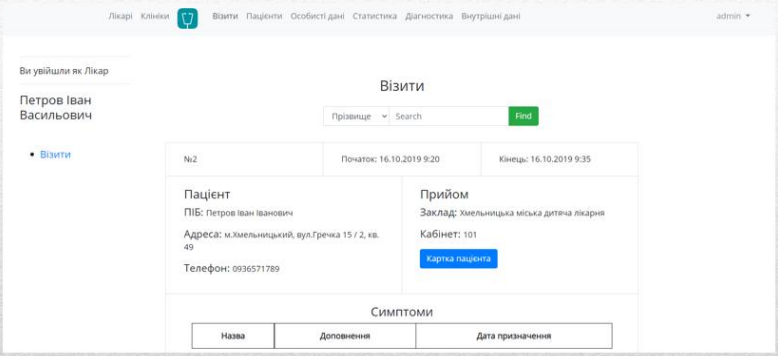
Демонстрація роботи програми

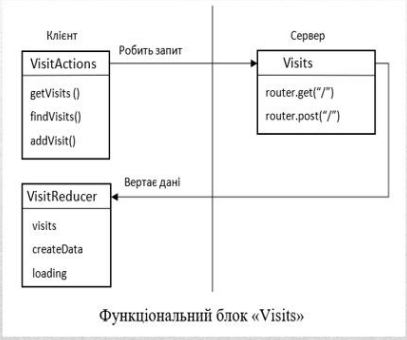




Функціональний блок «Diagnostics»

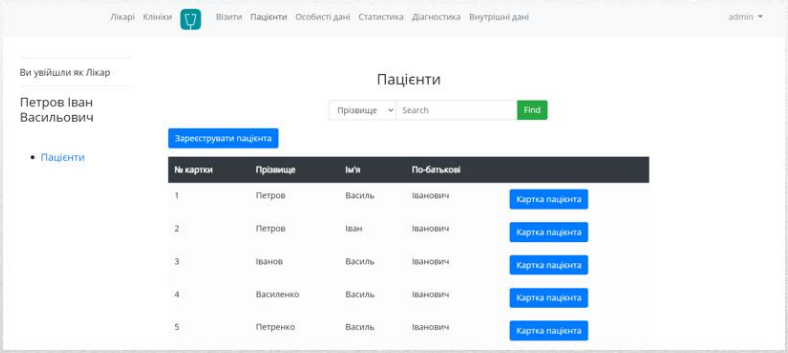
Демонстрація роботи програми

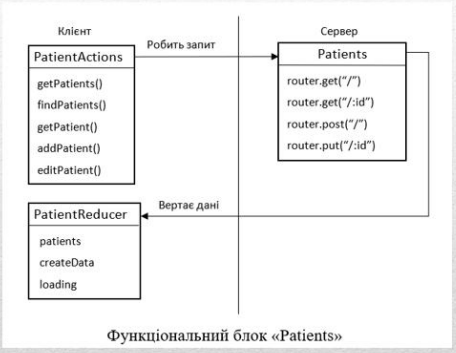




Функціональний блок «Visits»

Демонстрація роботи програми





Функціональний блок «Patients»

Висновки

В результаті виконання даної КРБ було розроблено автоматизовану систему допомоги в діагностуванні хвороб терапевтом на платформі JavaScript, використовуючи фреймворки Express, React, Redux і СКБД MySQL. Всі вимоги, поставленні до роботи, були виконані, а саме була створена база даних та застосунок, який її використовує. Створена система виконує наступні функції:

- ✓ Робота з відкритою інформацією (виведення інформації про лікарів та поліклініки, зареєстровані в системі).
- ✓ Робота з візитами (додавання симптомів пацієнта, діагнозів, направлень на процедури, рецептів на медикаменти).
- ✓ Робота з картками пацієнтів (додавання симптомів пацієнта, діагнозів, направлень на процедури, рецептів на медикаменти, редагування його особистої інформації).
- ✓ Робота з особистими даними (реєстрація в системі, авторизація, редагування даних профілю).
- ✓ Робота з внутрішніми даними системи (хвороби, симптоми, медикаменти, процедури, працівники, пацієнти, палати).
- ✓ Робота з допомогою в діагностуванні (отримання діагнозів по симптомах, додавання нових даних в систему діагностики).
- ✓ Робота зі статистикою роботи терапевта (по симптомах пацієнта, діагнозах, направленнях на процедури, рецептах на медикаменти).

Дана система може бути використана для автоматизації роботи терапевта поліклініки. Її можна вдосконалити, додавши можливість генерування направлень, можливість автоматизовано отримувати дані аналізів від лікарень, а також автоматизовану відправку направлень лікарням.

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ

Направляється студент Вишинський І. О. на захист дипломного проекту (роботи)
(прізвище, ініціали)

за спеціальністю 122 - Комп'ютерні науки

На тему: Автоматизована система допомоги в діагностуванні хвороб терапевтом

Дипломний проект (робота), рецензія і довідка про перевірку на плагіат додаються.

Декан факультету _____ САВЕНКО О.С.
(підпис) (прізвище та ініціали)

ДОВІДКА УСПІШНОСТІ

Вишинський І. О. за період навчання на факультеті програмування та компютерних і телекомунікаційних систем з 2017 по 2021 роки. повністю виконав навчальний план спеціальності з такими розподілом оцінок за:
національною шкалою: відмінно 43,75 %, добре 15,62 %, задовільно 40,62 %.
шкалою ЄКТС: A 34,55 %, B 18,18 %, C 7,27 %, D 14,55 %, E 25,45 %.

Методист факультету _____ _____
(підпис) (прізвище та ініціали)

ВИСНОВОК КЕРІВНИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ) ТА ОБГРУНТУВАННЯ ОЦІНКИ

Студент Вишинський І.О. представив свою кваліфікаційну роботу актуальному напрямку інформаційних технологій за сучасних умов, розробці системи допомоги в діагностуванні хвороб, яка здатна оптимізувати та прискорити діяльність сімейних лікарів та терапевтів. Всі завдання виконані, використано сучасні технології проектування і розробки.

Оцінка дипломного проекту (роботи) відмінно

Керівник дипломного прокту (роботи) _____ Мазурець О. В.
(підпис) (прізвище та ініціали)

"08" 06 2021 р.

ВИСНОВОК КАФЕДРИ ПРО ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Дипломний проект (роботу) розглянуто. Студент Вишинський І. О. допускається до захисту цього

Завідувач кафедри КНІТ
(назва)

_____ "08" 06 2021 р.
(підпис, прізвище, ініціали)

Wed Jun 09 14:04:40 EEST 2021, Петровський Сергій Степанович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 45.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 12%**

	Документ		Суммарное совпадение по Базе Данных	
ID: 92894 Название: Автоматизована система допомоги в діагностуванні хвороб терапевтом Добавлено в БД: 2021-06-09 Авторы: І.О. Вишинський Руководители: О.В. Мазурець Консультанты: Опоненты:	Символы	Лексемы	Символы	Лексемы
	48938	509	23499 (48%)	254 (50%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы
90054	Название: ЗВІТ з професійної практики Добавлено в БД: 2021-05-10 Авторы: Вишинський І.О. Руководители: Скрипник Т.К. Консультанты: Опоненты:	21991 (45.0%)	263 (52.0%)

UNICHECK

Ім'я користувача:
Кафедра КН

ID перевірки:
1008246124

Дата перевірки:
09.06.2021 14:55:31 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
09.06.2021 14:57:17 EEST

ID користувача:
100005671

Назва документа: **КРБ Вишинський 20210607 4 фінал повний Lite**

Кількість сторінок: **57** Кількість слів: **7668** Кількість символів: **58230** Розмір файлу: **5.63 MB** ID файлу: **1008318434**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

8.4%
Схожість

Найбільша схожість: **4.11%** з джерелом з Бібліотеки (ID файлу: **1008301751**)

3.04% Джерела з Інтернету	164	Сторінка 59
6.62% Джерела з Бібліотеки	51	Сторінка 60

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	4
Підозріле форматування	16 сторінок

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: *«Автоматизована система допомоги в діагностуванні хвороб терапевтом»*

Автор: *студент групи КН-17-2 Вишинський Ілля Олександрович*

Спеціальність: *122 – Комп'ютерні науки*

Освітня програма: *освітньо-професійна*

Науковий керівник: *доцент кафедри КНІТ Мазурець О.В.*

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	*відповідає*
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;

2) усі запозичення фрагментарні;

3) до запозичень входять фрагменти програмного коду, що на мають авторства і містять поширені конструкції;

4) серед запозичень знаходяться загальновідомі терміни, скорочення та визначення.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 8.4 % і адресується до першоджерел, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ О. В. Мазурець

Гарант ОП _____ О. В. Мазурець

Завідувач кафедри КНІТ _____ О. В. Бармак

Хмельницький національний університет
Кафедра комп'ютерних наук та інформаційних технологій

РЕЦЕНЗІЯ

на кваліфікаційну роботу бакалавра

студента *гр. КН-17-2 Вишинського Іллі Олександровича*

за темою *Автоматизована система допомоги в діагностуванні хвороб терапевтом*

1. Актуальність і значення теми *Наразі лікарі в медичних закладах надмірно навантажені рутинною роботою, такою як: реєстрація візиту пацієнта, зберігання даних про його здоров'я, заповнення його медичної картки. Крім того, їм потрібно призначати діагноз та певне лікування. Автоматизована система допомоги в діагностуванні дозволить автоматизувати роботу терапевта поліклініки, що підвищить його ефективність.*

2. Оцінка запропонованих моделей, підходів, алгоритмів, інформаційної складової та засобів розробки *Створена в роботі функціональна структура інформаційної системи та наведені шляхи автоматизації бізнес-процесів предметної області забезпечують можливість реєстрації пацієнта, реєстрації візиту, запису симптомів, медикаментів, процедур, схем лікування, призначених пацієнту, отримання статистики та отримання допомоги в діагностуванні хвороб, що є основним в роботі терапевта. При розробці інформаційної системи використано сучасні засоби програмування – мову програмування JavaScript з використанням фреймворків Express, React, Redux, а також СКБД MySQL.*

3. Оцінка розробленої інформаційної системи, її практична цінність та економічна доцільність *Розроблена інформаційна система дозволяє виконувати функції реєстрації пацієнта, реєстрації візиту, запису симптомів, медикаментів, процедур, схем лікування, призначених пацієнту, отримання статистики про візити, медикаменти, процедури, схеми лікувань, симптоми, отримання допомоги в діагностуванні хвороб. Результат роботи повністю відповідає поставленій задачі та може бути застосований за призначенням.*

4. Загальний висновок та оцінка *Всі поставлені завдання виконані повністю і на високому фаховому рівні. За своєю структурою, практичними цінностями, меті та вирішеними завданнями робота цілком відповідає вимогам вищої школи і вимогам, що пред'являються до освітньо-кваліфікаційного рівня «бакалавр», тому вважаю, що її автор Вишинський І.О. заслуговує присвоєння кваліфікації бакалавра з комп'ютерних наук.*

Робота заслуговує на оцінку «*відмінно*».

Рецензент *к.т.н., доц. каф. ТМІТ* Очеретний О.Ю.